



UNIVERSITÉ DE  
**SHERBROOKE**  
Faculté de génie  
Génie électrique et génie informatique

**UN SYSTÈME DE TRAITEMENT DE  
REQUÊTES SUR DES GRANDE BASES  
DE DONNÉES SPATIALES UTILISANT  
UNE NOUVELLE ALGÈBRE DE CARTE**

**A QUERY PROCESSING SYSTEM FOR  
VERY LARGE SPATIAL DATABASES  
USING A NEW MAP ALGEBRA**

Thèse de doctorat  
Spécialité : génie électrique

Seyed-Ali Firouzabadi

Sherbrooke (Québec) Canada

Octobre 2002



National Library  
of Canada

Bibliothèque nationale  
du Canada

Acquisitions and  
Bibliographic Services

Acquisitons et  
services bibliographiques

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file    Votre référence*

*ISBN: 0-612-86719-6*

*Our file    Notre référence*

*ISBN: 0-612-86719-6*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

**Canada**

# RÉSUMÉ

Dans cette thèse nous introduisons une approche de traitement de requêtes pour des bases de donnée spatiales. Nous expliquons aussi les concepts principaux que nous avons défini et développé: une algèbre spatiale et une approche à base de graphe utilisée dans l'optimisateur. L'algèbre spatiale est défini pour exprimer les requêtes et les règles de transformation pendant les différentes étapes de l'optimisation de requêtes. Nous avons essayé de définir l'algèbre la plus complète que possible pour couvrir une grande variété d'application. L'opérateur algébrique reçoit et produit seulement des carte. Les fonctions reçoivent des cartes et produisent des scalaires ou des objets. L'optimisateur reçoit la requête en expression algébrique et produit un QEP (Query Evaluation Plan) efficace dans deux étapes: génération de QEG (Query Evaluation Graph) et génération de QEP. Dans première étape un graphe (QEG) equivalent de l'expression algebrique est produit. Les règles de transformation sont utilisées pour transformer le graphe a un equivalent plus efficace. Dans deuxième étape un QEP est produit de QEG passé de l'étape précédente. Le QEP est un ensemble des opérations primitives consécutives qui produit les résultats finals (la réponse finale de la requête soumise au base de donnée).

Nous avons implémenté l'optimisateur, un générateur de requête spatiale aléatoire, et une base de donnée simulée. La base de donnée spatiale simulée est un ensemble de fonctions pour simuler des opérations spatiales primitives. Les requêtes aléatoires sont soumis à l'optimisateur. Les QEPs générées sont soumis au simulateur de base de données spatiale. Les résultats expérimentaux sont utilisés pour discuter les performances et les caractéristiques de l'optimisateur.

# ABSTRACT

The spatial databases are different to conventional databases, in data, query expression, indexing structures and consequently query optimization. While in conventional databases the objects are retrieved only based on their alphanumeric attributes, in a spatial database the objects are retrieved based on their shape, position and alphanumeric attributes. With the increase in size of the database (number of objects) and the complexity of the queries in spatial databases, efficient and optimized query processing becomes a critical issue.

In this thesis we introduce a query processing approach for spatial databases and explain the main concepts we defined and developed: a spatial algebra and a graph based approach used in the optimizer. The spatial algebra was defined to express queries and transformation rules during different steps of the query optimization. To cover a vast variety of potential applications, we tried to define the algebra as complete as possible. The algebra looks at the spatial data as maps of spatial objects. The algebraic operators act on the maps and result in new maps. Aggregate functions can act on maps and objects and produce objects or basic values (characters, numbers, etc.). The optimizer receives the query in algebraic expression and produces one efficient QEP (Query Evaluation Plan) through two main consecutive blocks: QEG (Query Evaluation Graph) generation and QEP generation. In QEG generation we construct a graph equivalent of the algebraic expression and then apply graph transformation rules to produce one efficient QEG. In QEP generation we receive the efficient QEG and do

predicate ordering and approximation and then generate the efficient QEP. The QEP is a set of consecutive phases that must be executed in the specified order. Each phase consist of one or more primitive operations. All primitive operations that are in the same phase can be executed in parallel.

We implemented the optimizer, a randomly spatial query generator and a simulated spatial database. The query generator produces random queries for the purpose of testing the optimizer. The simulated spatial database is a set of functions to simulate primitive spatial operations. They return the cost of the corresponding primitive operation according to input parameters. We put randomly generated queries to the optimizer, got the generated QEPs and put them to the spatial database simulator. We used the experimental results to discuss on the optimizer characteristics and performance.

The optimizer was designed for databases with a very large number of spatial objects nevertheless most of the concepts we used can be applied to all spatial information systems.

# ACKNOWLEDGEMENTS

I would like to express my endless gratitude to professor Ruben Gonzalez-Rubio, my supervisor, for his many suggestions and supports during this work.

Many thanks to the people of CRIM (Centre de Recherche Informatique de Montreal), Ying Li and others that I met and worked with during the early years of this research. I must also appreciate the fast and comprehensive services of the library of Sherbrooke University that was crucial to this work. Dr. Charles-Antoine Brunet provided  $\text{\LaTeX}$  class files implementing Sherbrooke University protocol of the thesis (in French), which I gratefully translated to English.

Also, I wish to thank the following: my wife Elham who joined my life towards the end of this work for her love and patience; and friends of Sherbrooke for their friendship and for sharing beautiful memories of Sherbrooke.

Sherbrooke, Quebec

Ali Firouzabadi

May 15, 2002

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	A perspective of the work . . . . .	3
1.1.1	The map model and algebra . . . . .	5
1.1.2	The spatial query optimizer . . . . .	7
1.1.3	Experiments . . . . .	7
1.2	Thesis contents . . . . .	8
<b>2</b>	<b>FUNDAMENTALS OF SPATIAL DATABASES</b>	<b>9</b>
2.1	Examples of spatial applications . . . . .	9
2.2	The differences to conventional DBMS . . . . .	11
2.2.1	Spatial relationships . . . . .	11
2.2.2	Spatial Operations and Functions . . . . .	13
2.2.3	Graphical presentation and user interface . . . . .	14
2.3	Spatial query languages . . . . .	16
2.4	Spatial data structures . . . . .	17
2.4.1	Data types in spatial databases . . . . .	18
2.4.2	Spatial modeling . . . . .	20
2.4.3	Indexing techniques in spatial databases . . . . .	21
2.4.4	Point indexing structures . . . . .	23
2.4.5	Rectangle indexing structures . . . . .	26
2.4.6	A general view of spatial access methods . . . . .	30

2.4.7	Graph modeling and indexing . . . . .	30
2.4.8	Evaluation of the spatial indexing structures . . . . .	32
2.5	Spatial query optimization . . . . .	33
2.5.1	Query transformation . . . . .	35
2.5.2	A set of optimization techniques . . . . .	36
2.5.3	A generic model for a query optimizer . . . . .	38
2.6	Spatial query processing architectures . . . . .	40
2.6.1	Separating spatial from aspatial queries . . . . .	40
2.6.2	Spatial filtering . . . . .	41
2.6.3	Extensible architectures . . . . .	42
<b>3</b>	<b>A BRIEF AND GENERAL LOOK AT THE WORK</b>	<b>45</b>
3.1	A new map algebra . . . . .	45
3.2	New query optimization and processing techniques . . . . .	46
3.3	The implemented system and experimental results . . . . .	48
<b>4</b>	<b>THE SPATIAL FORMALISM AND ALGEBRA</b>	<b>50</b>
4.1	Data elements . . . . .	51
4.2	Map operators . . . . .	53
4.3	Object operators . . . . .	56
4.4	Aggregate functions . . . . .	57
4.5	Predicates . . . . .	59
4.6	Examples . . . . .	60
4.7	Query evaluation graph . . . . .	61
4.7.1	Graph elements . . . . .	62
4.7.2	Example graphs . . . . .	63
4.8	Transformation rules . . . . .	65
4.9	Conclusion . . . . .	67

<b>5</b>	<b>THE SPATIAL QUERY OPTIMIZER</b>	<b>70</b>
5.1	Primary interpretation . . . . .	71
5.2	QEG generation . . . . .	72
5.2.1	Graph construction . . . . .	72
5.2.2	Graph transformation . . . . .	74
5.3	QEP generation . . . . .	76
5.3.1	Spatial predicate evaluation . . . . .	80
5.4	Conclusion . . . . .	86
<b>6</b>	<b>EXPERIMENTAL RESULTS</b>	<b>88</b>
6.1	The implemented query optimizer . . . . .	88
6.1.1	Random query generator . . . . .	89
6.1.2	QEG generator and optimizer . . . . .	89
6.1.3	QEP generator and optimizer . . . . .	90
6.1.4	Simulator of spatial primitive operation processor . . . . .	91
6.2	Experiments . . . . .	92
6.3	conclusion . . . . .	94
	<b>CONCLUSION</b>	<b>96</b>
<b>A</b>	<b>A BNF Grammar For The Spatial Algebra</b>	<b>99</b>
<b>B</b>	<b>Experimental Results</b>	<b>103</b>
B.1	Average estimated costs and improvements . . . . .	103
B.2	Example randomly generated queries and their QEP before and after applying the optimization techniques . . . . .	106
	<b>BIBLIOGRAPHY</b>	<b>112</b>

# LIST OF FIGURES

2.1	Topological relationships . . . . .	13
2.2	Indexing structures . . . . .	22
2.3	A two dimensional region quad-tree . . . . .	24
2.4	A two dimensional point quad-tree . . . . .	25
2.5	A two dimensional k-d tree . . . . .	27
2.6	The structure and planar representation of R-tree . . . . .	28
2.7	The structure and planar representation of $R^+$ -tree . . . . .	29
2.8	Spatial indexing methods . . . . .	31
2.9	Basic structure of a query optimizer . . . . .	34
2.10	Generic model for a query optimizer in database systems . . . . .	39
2.11	An example architecture of a query processor in spatial databases . . . . .	41
2.12	Optimizer generator paradigm . . . . .	43
3.1	Our optimizer in a spatial DBMS . . . . .	47
3.2	Potential use of our work in combination with other approaches . . . . .	48
4.1	A spatial database . . . . .	52

4.2	Query graph elements . . . . .	62
4.3	An equivalent graph for Join (iff $out = x_1$ ) . . . . .	63
4.4	QEG for Example 1 . . . . .	63
4.5	QEG for Example 2 . . . . .	64
4.6	QEG for Example 3 . . . . .	65
5.1	Spatial query optimization system . . . . .	71
5.2	QEG for Example 2 after transformation . . . . .	79
6.1	Cost improvement for queries with nesting level = 2 . . . . .	93
6.2	Cost improvement for queries with nesting level = 4 . . . . .	94

# LIST OF TABLES

2.1	Spatial query languages . . . . .	18
4.1	Spatial operators . . . . .	58
5.1	Spatial predicate approximation . . . . .	86

# ACRONYMS AND GLOSSARY

QEP	Query Evaluation Plan
QEG	Query Evaluation Graph
QL	Query language
GUI	Graphical User Interface
GIS	Geographical Information System
GPS	Global Positioning System
VLSI	Very Large Scale Integrated circuits
ISO	International Standard Organization
SQL	A modified version of Structured Query Language of IBM, introduced in 1986 by database committee of ANSI (X3H2)
SQL3	The latest version of SQL that covers spatial databases and object oriented databases.
DBMS	Database Management System
WWW	World Wide Web
BNF	Backus Normal Form (also some people regard it as Backus-Naur Form). A notation used to specify the syntax of programming languages or command sets.
QBE	Query By Example.
QUEL	The query language used by the database management system INGRES.
ANSI	American National Standards Institute.

RWO	Real World Object
MBR	Minimum Bounding Rectangle
CNF	Conjunctive Normal Form. A logical formula consisting of a conjunctions of disjunction where no disjunction contains a conjunctions.
DNF	Disjunctive Normal Form. A logical formula consisting of a disjunction of conjunctions where no conjunction contains a disjunction.
OODB	Object Oriented Database
CAD	Computer Aided Design

# Chapter 1

## INTRODUCTION

Spatial database systems have had an increasing importance and role in computer based applications since the 1960's due to the dramatic improvement and progress of computer systems [SEAI, 1990]. Applications that are considered in this domain cover a vast area, from Geographical Information Systems (GIS) [LAXTON, 1996] to VLSI design. In all of these systems, storing and retrieving multi-dimensional data based on queries from users, is the main goal. Users of these systems put their queries to them by using a Query Language [JARKE, 1989b](QL) or Graphical User Interface (GUI). Then a query processing system will be responsible for processing the queries.

Designing an efficient query processing system for spatial databases has been the subject of research for many researchers in industry, research centers and standard committees and institutes for a long time [RISHE 95, SAMET, 1995, ISO/TC 211, SQL Home Page]. Many spatial query languages and query processing techniques have been introduced. Some of them are extensions to previous conventional database systems to use their abilities for the aspatial (non-spatial) part of data, and some of them are new systems designed specifically for spatial data management. By far, we can summarize the open and important issues in spatial databases as following:

1. Indexing techniques to retrieve data based on the spatial constraint and criteria: There are many indexing techniques, introduced for spatial databases. The characteristics of the database and its applications determine which indexing techniques are more appropriate. New indexing techniques are still being introduced [SELLIS et al., 1997].
2. Data structures for spatial data: For example for a GIS with a huge amount of images, to retrieve the images based on the contents of them, new indexing methods and data structures should be used. These data structures should keep information about the contents (semantic) of each image. Also, new compression techniques should be introduced to minimize the storage space.
3. New languages to support the new data types and spatial queries: Almost every existing spatial information system has its own language (a lot of them are SQL based). SQL3 standard language is being introduced by ISO. In addition to textual languages, visual languages and Graphical User Interface are other ways of interaction with spatial information systems.
4. The formalism and the algebra: This is necessary for query expression, query transformation and query optimization procedures. The goal is to achieve an algebra that is sound and complete. It also must provide effective transformation rules to have the simplest equivalents of the query out of query transformation phase.
5. New query processing and optimization techniques: It is the most important part of the query processing procedure in a DBMS. A query optimizer design depends on the database and the application characteristics and it uses many optimization techniques through passing several steps of optimization procedure [JARKE, 1984, IONNIDIS, 1996]. Some optimization techniques are general to every kind of databases and some are particular to a special database. A query

optimizer usually uses a set of general and particular approaches.

Other open issues some of which may be related to the above issues are: Spatial deduction and reasoning in deductive spatial databases [LU et al., 1995], Spatial data mining [WEI et al., 1997], Clustering methods in spatial databases [SANDER et al., 98] [ERWIG et al., 1997] [ESTER et al., 1998] [SHEIKHOLESAMI et al., 2000], Spatial database inter-operability or Distributed spatial databases, Nearest neighbor query, Spatial join processing strategies [THEODORIDIS et al., 1998] [IBRAHIM et al., 1996], Selectivity estimation in spatial databases, Spatial databases on World Wide Web (WWW) [BRABEC et al., 1998a] [BRABEC et al., 1998b] [SMITH, 1998], Spatial database applications in: traffic and transport control [HILLMAN, 1997] [BRINKHOFF, 1999], biology, mobile communication, etc., Processing fuzzy queries in spatial databases, and Topological queries in spatial databases [SEGOUFIN et al., 1998]. Some of the issues like Spatial databases on the WWW are very new and in fact were introduced only a few years ago.

In this work we will mostly deal with two main issues: defining an algebra and designing a query optimizer for spatial databases. These are quite fundamental to the efficiency of every spatial application and we still don't have a model that is totally specific to spatial databases. The existing models are all extensions of non-spatial databases.

## 1.1 A perspective of the work

To explain our work through the thesis we will use examples of spatial database applications. Here we bring three examples. Then we see how our work is used to express and process the queries in the examples.

**Example 1** In a GIS that contains the map of a country one example query can be: Find cities with a population bigger than 10000 situated on a 50 km neighborhood of a fault line.

**Example2** In a CAD database containing the map of a building one example query can be: Show rooms with at least one smoke detector and one fire-fighting box in its 20m approach. Evaluate the query in the window of interest.

**Example3** In a database containing the neural structure of the brain (A potential future application of spatial databases in biology) find neurons connected to a maximum 10 other neurons in a specified window.

There are two sets of questions concerning the examples:

1. How to organize the data in the applications? How to express the queries in these examples with a common formalism that is sound and complete<sup>1</sup>? Can the proposed formalism express queries in other similar applications as well? What should be the operators? Should they be primitive or not? What are the consequences?
2. How to evaluate the written queries in the most efficient way. Are there other equivalent expressions that are easier to evaluate? Which predicate should be tested first? Is it helpful to use approximation to prune the search domain? How?

The answer to the first set of questions will be a model and an algebra. The answer to the second set of questions will be an optimizer.

From early works in this domain until now many different models have been proposed most of which are extensions to the existing models for non-spatial databases. Relational,

---

<sup>1</sup>Although introducing a high level spatial query language and user interface is a challenge for different standard committees but it is not as fundamental and important as formalism and algebra. A good formalism and algebra is essential to the performance in database design and to the efficiency in query interpretation.

object-relational, object-oriented, deductive object oriented, and deductive relational, models are proposed with an extension to support spatial data [SHEKHAR et al., 1999]. A new model and algebra that is fully dedicated to spatial databases has not been proposed yet.

### 1.1.1 The map model and algebra

In our approach to efficient query processing in spatial databases we propose a new model for spatial data, named map model. It looks at spatial data as objects in different maps. The objects have position and shape in addition to the optional non-spatial data. This way of grouping the data is neither relational nor object oriented and it is not an extension to the existing models. All maps share the same universal space and coordination. They don't have hierarchy. As we will see in the design of our spatial database the definition of maps is based on the type of potential user queries rather than concepts like primary or foreign keys of relational model. This lets us to use efficient techniques in query optimization. The algebra is based on our model (map model) and as in other algebras we define it in three parts:

- Data elements: We introduce map, spatial object, and atomic values as data elements of our algebra.
- Operators: We introduce set operators, Spatial *Join*, *Select*, *Window* and *Point* operators. Because the data both in our model and relational model is grouped in sets of objects (map) or sets of tuples(relation), we have many similar operators (set operators) to relational algebra. Each operator do operation on map(s) and produces a new map as result. Aggregate functions receive maps or objects in input and give map, object, or atomic value as result. Some operators use predicates. To express spatial predicates, we define a set of spatial relationships.

- Equations and transformations: We introduce equations in algebraic expressions (transformation rules) and bring the proof for some of them. They will be used in query optimization.

For example we will see that Selection and Window operators have the mutation property and it is more efficient to do the Window operation before the Select operation.

Now we look at the above three examples in our map model and see how to express them in map algebra.

**Example 1** The data in this application should be in several maps. Let assume that we have one map for cities (named City) and one map for fault lines (named Fault-line). The example query can be expressed in two operations: Select and Join. First we do a Select on City and then Join the result with Fault-line meeting neighborhood predicate.

**Example 2** Let us assume that we have one map for rooms (named Room), one for fire fighting box (named Fire-Fight) and one for smoke detectors (named Smoke-Detect). The query can be expressed using two Join operations and one Window operation. First we do a Join on Room and Fire-Fight. Then we do Join the result and Smoke-Detect. Then we do Window on the result.

**Example 3** This is a special example. It is at the same time simple and complex. The query in this example can be expressed by only one Select operation on one map containing the neurons (named Brain). But the predicate in the Select operator will be itself a sub-query with a Join operation.

We will review these examples more extensively in next chapters.

### 1.1.2 The spatial query optimizer

The main application of our algebra is in the query optimizer. The optimizer receives the query in algebraic expression and produces one efficient QEP (Query Evaluation Plan) through two main consecutive blocks:

- QEG (Query Evaluation Graph) generation. The input to this block is the algebraic query and the output is a QEG. Using the transformation rules expressed in our algebra we rewrite the query in a more efficient expression. Then its graph equivalent is constructed using the graph components we defined. The achieved graph is then transformed into a more efficient equivalent through some transformation rules. We recognized that QEG is quite useful for spatial query processing as they let us apply the filtering and refinement approach in an efficient way (as we will explain it in chapter 5.).
- QEP generation. In QEP generation we receive the efficient QEG and do predicate ordering and approximation and then generate the efficient QEP. The QEP is a set of consecutive phases that must be executed in the specified order. Each phase consist of at least one or more primitive operations. All operations with the same distance from the end point of QEG (query result) can be in the same phase. All primitive operations that are in the same phase can be executed in parallel. A cost model is introduced to calculate the relative cost of a combinational predicate based on the order of evaluation of the predicates, then choose the least expensive order. We will explain this in chapter 5

### 1.1.3 Experiments

To evaluate our proposed solution for efficient spatial query processing (the spatial query optimizer) and extract its characteristics we implemented the optimizer, a randomly

spatial query generator and a simulated spatial database. The query generator produces random queries for the purpose of testing the optimizer. The simulated spatial database is a set of functions to simulate primitive spatial operations. They return the cost of the corresponding primitive operation according to input parameters. We put randomly generated queries to the optimizer, got the generated QEPs and put them to the spatial database simulator and got the total cost of evaluation for each query. We used the results to discuss on the optimizer performance versus different parameter such as the number of nested operators, number of objects in the maps and number of maps. We also compared the query processing cost for different optimization techniques.

## 1.2 Thesis contents

The rest of the thesis is organized as following: We survey the state of the art for the subject in Chapter 2. Then we present a general look of the accomplished work in chapter 3. It gives a quick perspective of the work. In next chapters we go further in details of the accomplished work. Chapter 4 introduces the invented algebra in details. The optimizer, its structure and its different modules are presented in chapter 5. Chapter 6 discusses on the optimizer characteristic and performance based on the experimental results. Finally we bring a summary and conclusion of the work in the CONCLUSION chapter.

There are two appendices to the thesis. Appendix A is a BNF grammar of the algebra. It is useful for how to read the queries written in algebraic expression. Appendix B contains experimental results.

## Chapter 2

# FUNDAMENTALS OF SPATIAL DATABASES

In this section we will present the main issues in spatial DBMS. Some of these issues are subjects of advanced researches in database management systems but here, we will discuss them briefly to have only a general idea about spatial information systems.

### 2.1 Examples of spatial applications

Although there is a wide range of different spatial applications, they use almost the same tools and concepts to manage the spatial data. To have a background for further discussions in this document we bring an application and its example queries.

One famous application of spatial databases in the context of GIS is to put the map of a city with all of its objects (e.g. roads, schools, hospitals, electric network and its elements) in a computer database to handle the queries in the easiest and fastest way. Such a comprehensive database can be in fact a set of many smaller maps (e.g. road map and electric network map). The example queries on this database could be:

**Example 1** Find the nearest power station to client A.

This is a query based on the geometrical part of objects. The nearest relationship is a neighborhood relationship. The result of the query could be the names or other attributes of the objects found.

**Example 2** Find the number of transformers in area B.

In other words the query is to find all objects of a specified class (transformers) that qualify the predicate: intersect with area B. As we see, the query is about the geometrical part of objects and the restriction predicate is based on topological relationship.

**Example 3** Find one way roads that intersect “Sainte-Catherine” Street.

The restriction predicate in this query is based on both alphanumerical and geometrical part of the objects. Being one way is an alphanumeric attribute and intersecting other object (“Sainte-Catherine” Street) is a topological relationship based on the geometrical part of the objects.

**Example 4** Find roads that intersect “Sherbrooke” Street and are longer than 1 km.

This query, as in example 3, is based on the alphanumerical (length of the road) and geometrical attribute of objects.

**Example 5** Find parking lots that are at most 500m from “Queen Elizabeth” hotel and on its west side.

The restriction predicate in this query is based on order relationship (being west of “Queen Elizabeth” hotel)

**Example 6** Find the minimum length path between “Edouard Montpetit” high-school and “Saint-Luc” hospital.

This query is done by using an aggregate function. The inputs of the function are two objects and the output is the minimum path between them that can be declared as a new object or a combination of several objects (pieces of the roads in the path).

In the above examples we tried to have queries based on both geometrical and alphanumerical part of spatial objects. We also tried to have topological relationships, order relationships, neighborhood and special functions and operations in our queries. We will talk about these concepts in the coming section.

## 2.2 The differences to conventional DBMS

Spatial databases are different from conventional or relational databases in data types, relationships<sup>1</sup>, operations and presentation of query results. Each object in a spatial database, in addition to aspatial attributes that are integers or string of characters, must have a geometrical attribute for presenting spatial characteristics or position of the object.

In this section we just talk about relationships, operations and presentation of query results.

### 2.2.1 Spatial relationships

While relationships in the conventional QL are  $<$ ,  $>$ ,  $=$  or a combination of them, we encounter new complex ones in spatial QL. These relationships describe the situation

---

<sup>1</sup>Relation in database has a different meaning to relationship in query language

and position of an object to the others and they are related to the geometrical characteristics of each object. We can categorize them in two categories: topological and order relationships

**Order Relationships** Here is a list of spatial order relationships that can be supported by the QL of a spatial DBMS:

- left of/right of
- beside (alongside, next to)
- above (over, higher than)
- below (under, lower than)
- behind (to the back of)
- in front of
- near/far
- between

Some spatial QL support all of them and in some cases, only a subset of them are supported.

**Topological Relationships** Although neighborhood relationships or topological relationship can be expressed by use of geometrical formulas, putting them in the syntax of QL and considering them as part of the language dictionary, releases users from cumbersome geometrical calculations and puts the language in a higher level.

These relationships are shown in figure 2.1.

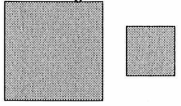
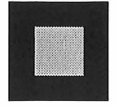
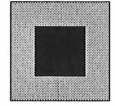


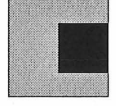

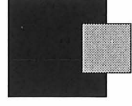
disjoint 	contains 	inside 	equal 
meet 	covers 	covered by 	overlap 

Figure 2.1: Topological relationships

A spatial query can be a combination of subqueries based on the above relationships (topological and order) and conventional relationships ( $<$ ,  $>$ ,  $=$ ). It can be expressed and processed based on an algebra (a new spatial algebra) as for queries based on conventional relationships [PAREDAENS, 1995].

### 2.2.2 Spatial Operations and Functions

In spatial DBMS we see a new set of operations in addition to the conventional operations (in an algebra operations are represented as operators and aggregate functions). While in conventional QL, we had only some simple operations such as mean square value or mean value that were acting on the character or integer attribute of objects, here we have operations that act on the geometrical part of the data. Functions that calculate the perimeter, area and volume of the objects are examples of spatial operations.

Sometimes for a special application we may need a complex function. The operation that acts on two objects in a city map and extracts the optimum path between them according to dynamic data (e.g. traffic status) is an example for such type of spatial

operations.

We show how the relationships and operations can be used in typical QL through an example.

**Example** Suppose that we have a spatial database that holds a complete map of Montreal with all places specified on it. To find all hotels on Sherbrooke street that have more than 50 rooms and have an area greater than 2000  $m^2$ , we need the relationship: *meet*<sup>2</sup> and the operation: *area* in addition to conventional ones.

```
SELECT name FROM building, road WHERE
```

```
road.name = "Sherbrooke" and
```

```
road.geometry MEET building.geometry and
```

```
building.type = "hotel" and
```

```
AREA building.geometry > 2000 and
```

```
building.room > 50
```

### 2.2.3 Graphical presentation and user interface

Results of queries to spatial database are objects that have both spatial and aspatial data. We can display the aspatial part in tabular form. But to display the spatial part,

---

<sup>2</sup>This example does not use any specific language, but the statements are written as SQL based language

we need a graphical media. Because of the importance of this graphical environment, we must put the power of management in the hands of the user as much as possible. Normally this is done by using a language. This language is added as a complementary but distinct part to the main query language (Graphical Presentation Language or Graphical Representation Language ) [EGENHOFER, 1994].

We can categorize characteristics of such a user interface as following:

**Display mode.** This says how to combine the results of queries. The user must be able to add the new results over the previous ones, start a new result screen, intersect the current and previous results or highlight important parts of query results.

**Visual variables.** These variables (colors, patterns and symbols) determine the color, pattern of regions and the symbols that represent objects. Thus different users by determining these variables in their own manner can see one query result in different manner. Each one according to her/his interest.

**Scale and window.** By setting the window, users can determine the area that they are interested in and they want to see. They can also set the scale of presented area by setting the scale variable.

**Context.** Interpreting the results is highly dependent on the context. For example, without showing the borders of a state or country, users can't have any estimation about the situation of a city from a single point that represents the city.

Also in spatial DBMS it is preferred to integrate query interface and result presentation in one interface rather than separate interfaces. For example while you look at the map of a city that is generated as result of previous query you simply click on objects and side menu (may be constraints) to make another query (query by pointing) or modification to graphical presentation [LEE, 1995].

## 2.3 Spatial query languages

While extending conventional query languages for special purposes (e.g. spatial or temporal databases) is easier than introducing new ones, they usually do not have the efficiency and power of new query languages that are designed specifically for a special purpose.

In designing, extending or choosing a query language for spatial databases there are some basic parameters that are usually considered and of course there is a trade off between them: type of data, user friendliness, supported spatial relationships and speed of query processing are among those factors.

From several query languages for relational and conventional databases such as QUEL, QBE, and SQL, one of them (SQL) received the most attention by users and DBMS designers [BCS, 1981]. The standard language that was introduced in 1986 by ANSI database committee (X3H2) was a modified version of Structured Query Language (SQL) of IBM. Very soon it was accepted as an international standard by ISO. Now, almost every DBMS supports this language and all DBMS users are familiar with it. We can even see special types of database systems (e.g. temporal DBMS or spatial DBMS) use a SQL based syntax and structure as their query language [EGENHOFER, 1994, DE FELICE, 1992]. In addition to the efficiency of SQL, the main reason for this decision is that they want their users, who are mostly familiar with SQL, to be readily able to use the new language.

Based on the above discussion, the existing query languages for spatial databases can be categorized in two categories:

1. Extensions to conventional languages
2. Other (autonomous) languages

The first group consists mostly of the extensions of three languages: SQL, QUEL and QBE. In table 2.1 we have included the most important ones. Some of these languages are still under progress (e.g. SQL3). Some of them were just proposed by a research center and there is no application that has been based on them (e.g. spatial SQL) and some of them are used by an existing DBMS.

The second group consists of languages that have been proposed specifically to support spatial or object oriented databases. Many of them are visual<sup>3</sup> query languages (Ciglas, MVQL, EVA, Graqla).

The reason for so many different languages may be that each DBMS has its own characteristics and will be more efficiently supported by a dedicated language that is designed according to these characteristics. Nevertheless the efforts to have a standard language have been maintained during the evolution of the different types of DBMS. SQL3 is the latest standard language for object oriented databases in general hopefully to be applied for spatial databases in particular. SQL3 was supposed to be ready by 1996 but finally in year 1999 it was released as SQL:99 and it is yet to be accepted as ISO standard. SQL:99 is a query language for object oriented databases with the provision of user defined data types and methods to support spatial databases.

## 2.4 Spatial data structures

Data structures and search algorithms to retrieve data (indexing mechanisms) are major determinants of the overall performance of a DBMS. In spatial DBMS, in addition to conventional indexing structures that support alphanumeric part of data, we need some mechanisms to support geometric parts of data. These indexing structures are used to keep the object itself and the information about spatial order and position of the object

---

<sup>3</sup>A query language is said to be visual whenever the semantics of the query is expressed by drawing.

SQL Based	QUEL Based
SQL3 ODMG93 PSQL [ROUSOPOILOS, 1985] MAPQUERY [FRANK, 1982] Spa SQL Spatial SQL [EGENHOFER, 1994] Object SQL [OAKLEY, 1994] GSQL [HWANG, 1994] GISQL [COSTAGLIOLA, 1995] GEOQL [OOI, 1991]	GEO-QUEL [BERMAN, 1977] GEM POSTQUEL [STONEBRAKER, 1987]
QBE Based	Other languages
QBPE [CHANG, 1981] GEOBASE [BARRERA, 1981] PICQUERY [JOSEPH, 1988]	GeoSAL PROBE [ORENSTEIN, 1988] LOREL Ciglas [CALCINELLI, 1994] MVQL [OAKLEY, 1994] EVA [GOLSHANI, 1992, GOLSHANI, 1994] Graqla [JUNGERT, 1993]

Table 2.1: Spatial query languages

(or the shape of the object). As in the conventional databases the main goal of indexing structures is to support the corresponding queries (here spatial queries) in the most efficient way.

Before presenting and evaluating the indexing techniques in spatial databases, we first analyze the basic data types in these databases.

#### 2.4.1 Data types in spatial databases

In every kind of spatial applications there are basic data types that are used as the components of the database objects. As in object oriented databases the data objects of the spatial database are a combination of these basic data types. According to their characteristics and behavior we categorize them in three groups as following:

1. **Alphanumeric:** As in the traditional databases this type represents a string of characters that can be number or a name. For example in a database that holds the information of road signs in a city the name of the road signs is the alphanumeric part of the information about each sign and must be kept in a alphanumeric data field. Another example can be the population of the cities in a database that keeps the geographical information and the map of a country.
2. **Spatial:** This type represents the spatial information (position and shape). It is used to keep the spatial specification of the objects. In a database which contains points, lines and polygons, data fields that keep the position of the point in the space or position and direction of lines are of spatial type. Although spatial type is a set of numbers (e.g.  $x$  and  $y$  that represent the coordination of a point in a 2D space are real numbers) but is different to alphanumeric in the operations and relations that can be defined on them. The relationships for alphanumeric type are  $<$ ,  $>$ ,  $=$ ,  $\leq$  and  $\geq$ , while for spatial type we have topological and order relationships.
3. **Graphical and multimedia:** In many spatial database systems we have more detailed and advanced information about spatial objects like, audio and video data, images and legends. This information is represented by types different to alphanumeric and spatial. They are different to spatial types because spatial relationships and operators are not defined for them (Although in some cases we can extract spatial characteristics of an image and put it in spatial type but there might be information that are not spatial, like the color of pixels and resolution). They are different to alphanumeric types by the same reason. Because these types can keep a lot of information about the objects, there are techniques to extract the semantic of them (e.g. images and videos) and index the objects based on the extracted characteristics [GOLSHANI, 1994, VASSILAKOPOULOS, 1995].

Image processing, innovative indexing techniques based on the semantic of the data and data compression techniques are issues that are relevant to these types and are subjects of research in MMIS (Multimedia Information Systems).

### 2.4.2 Spatial modeling

Real world objects are computerized and stored in databases by use of spatial modeling. The spatial objects and their corresponding spatial modeling could be categorized as following:

1. Points (Zero dimensional objects in a  $n$  dimensional space) are modeled by:

- Cartesian coordinates
- Polar coordinates

2. Lines (One dimensional objects in a  $n$  dimensions space) are modeled by:

- Two points of the line
- One point and the direction of the line

A polygon or polyline can be modeled as a region or as set of points and lines.

3. Regions ( $k$  dimensional objects in a  $n$  dimensional space) are modeled by:

- Region quad trees: will be explained in section 2.4.4.
- Rectangle approximation: The smallest rectangle that contains the object is referred by MBR (Minimum Bounding Rectangle). Some systems use other polygons rather than rectangles. These approximation of the objects are used to prune the search space during query processing. More details are brought in section 5.3.1

- Raster model: In this model spatial objects are represented by a set of finite points (raster points) they contain. Regarding that there is a finite number of predefined raster points, this model will always represent an approximation of the object.
- Pizza (Peano) models: In this model we walk in the region on a predefined filling curve and report in every step if the point is black or white. Obviously the approximation depends on the filling curve and the size of the steps. The advantage to raster model is that the filling curve will sweep regions of geometrical information with a sharper approximation.
- Spaghetti (Vectorization) model: In general in this model the information in  $n$  dimensional space is represented by  $m$  dimensional hyper-spaces where  $m < n$ . For example in a two dimensional space polygons are represented by lines and graphs. Lines and graphs are represented by points, and points are represented by their coordinates.
- Polynomial model: In this model each object is expressed by its algebraic expression. For example a disk can be represented by:  $(x_1 - 2)^2 + (x_2 - 3)^2 < 25$ . Obviously only specific shapes (shapes that can be expressed by geometrical formulas) can be represented this way.

A detailed explanation of these models can be found in [LAURINI, 1992].

### 2.4.3 Indexing techniques in spatial databases

An indexing structure in the conventional databases is a structure that indexes the data based on one of the data fields. The structure links each occurrence of the field to the corresponding record or object. The most famous example of these structures are B-trees. A spatial indexing structure indexes the objects based on their spatial positions

(Figure 2.2). In other words the structure links each database object to its spatial position.

What should be reminded is that indexing structures are different to spatial modelling structures. The difference is that as we see in figure 2.2 the indexing structures are used to accelerate the search operations in performing a query. While the spatial modelling structures just represent the spatial objects in a database (a digital representation of Real World Object). In other words the indexing structures are used to index the data resulting from spatial modeling structures.

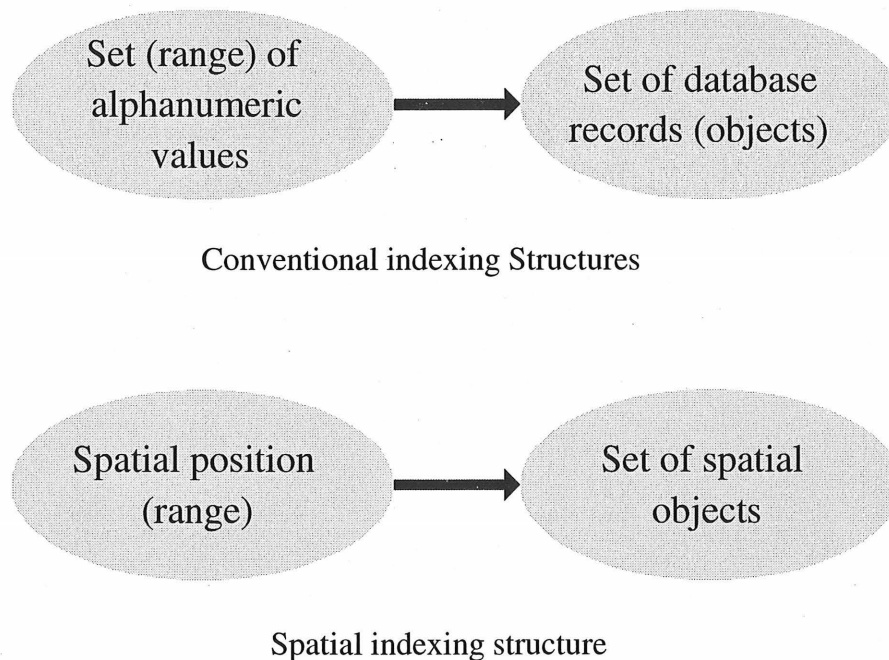


Figure 2.2: Indexing structures

Indexing structures are used in the final steps of interpreting a query to perform the basic simple queries resulted from breaking a higher level and more complex query. For example all kinds of spatial queries (neighborhood, topological and order relationship) could be interpreted by breaking down to only two simple queries: point query and window query. The functions that perform a simple spatial query based on the indexing

structures are less complicated than the structures themselves but the procedure of breaking complex and high level queries to these simple queries can be difficult and from then the query optimization becomes important.

The spatial indexing structures can be categorized in two types (depending on the objects they index):

1. Point indexing structures
2. Rectangle indexing structures

In each of the above categories a lot of indexing structures have been introduced, but they are mostly expansions and different versions of some basic ones. We will have a short look on these basic structures of each category.

#### 2.4.4 Point indexing structures

There are many point indexing structures. Here we bring the most famous ones. Although the Region Quad-Tree is not a point indexing structure but many point indexing structures use the same concept.

**Region Quad-Trees** [KLINGER, 1971] In Region Quad-Tree that was proposed by Klinger, the region is splitted to disjoint standard sized quadrants in a repetitive manner. The decomposition continues only on quadrants that are partly filled with the region or image. Quadrants that are empty or fully filled are the end points. Decomposition can be continued until desired resolution.

This technique of presenting images is very suitable for image processing purposes. Nevertheless points in a spatial database can be represented by this technique, if we consider

each point as a least sized filled quadrant according to our desired resolution (see figure 2.3).

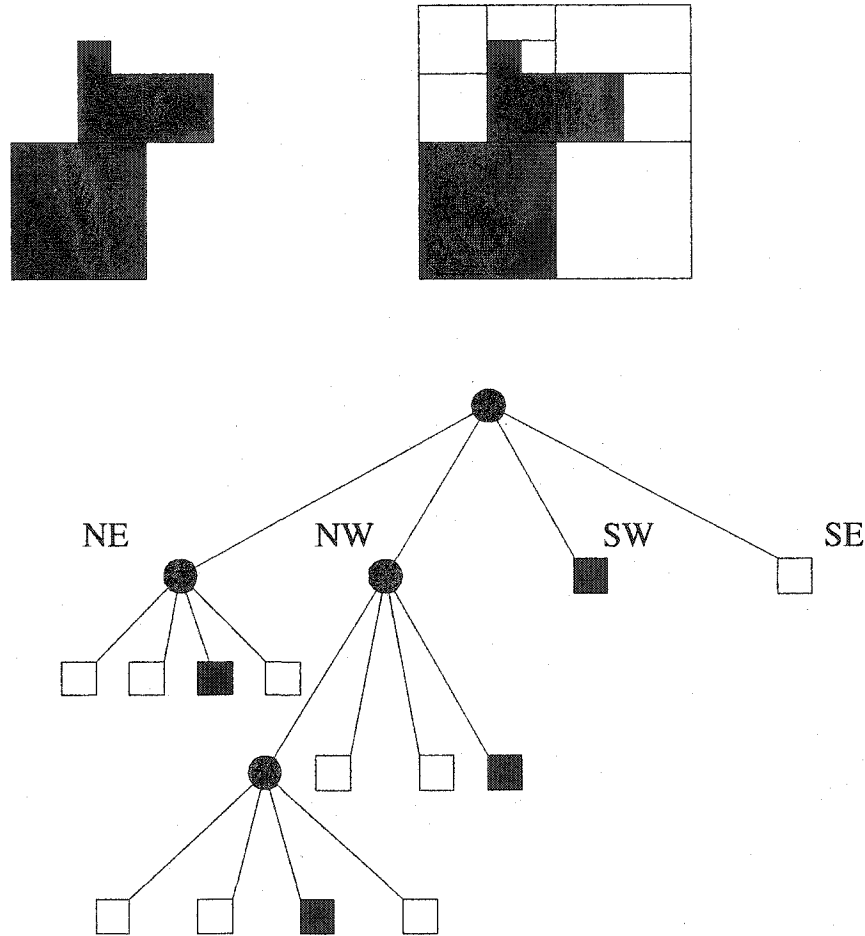


Figure 2.3: A two dimensional region quad-tree

**Point Quad-Trees** This structure was proposed by Finkel and Bentley [FINKEL, 1974] to represent points in a  $k$  dimensional space. Each node of the tree represents a point object. Each node may have a maximum of  $2^k$  branches. Thus in a 2-dimensional space each node of Point Quad-Tree has 4 sons to represent 4 directions: NW, NE, SW and SE (see figure 2.4). The branching continues until all point objects are represented.

This structure is suitable for exact match searches. Nevertheless range search queries

are well supported by it.

There are three other versions of this structure: Pseudo Quad-Tree, MX (Matrix) Quad-Tree and PR (Point Region) Quad-Tree. To improve some characteristics of this structure and to make it more dynamic, Overmars and Leeuwen [OVERMARS, 1982] introduced Pseudo Quad-Tree. In this method, internal nodes are not data points. They are arbitrary nodes that are chosen to divide efficiently the subquadrants. This simplifies deletion of data points in comparison to ordinary Quad-Tree.

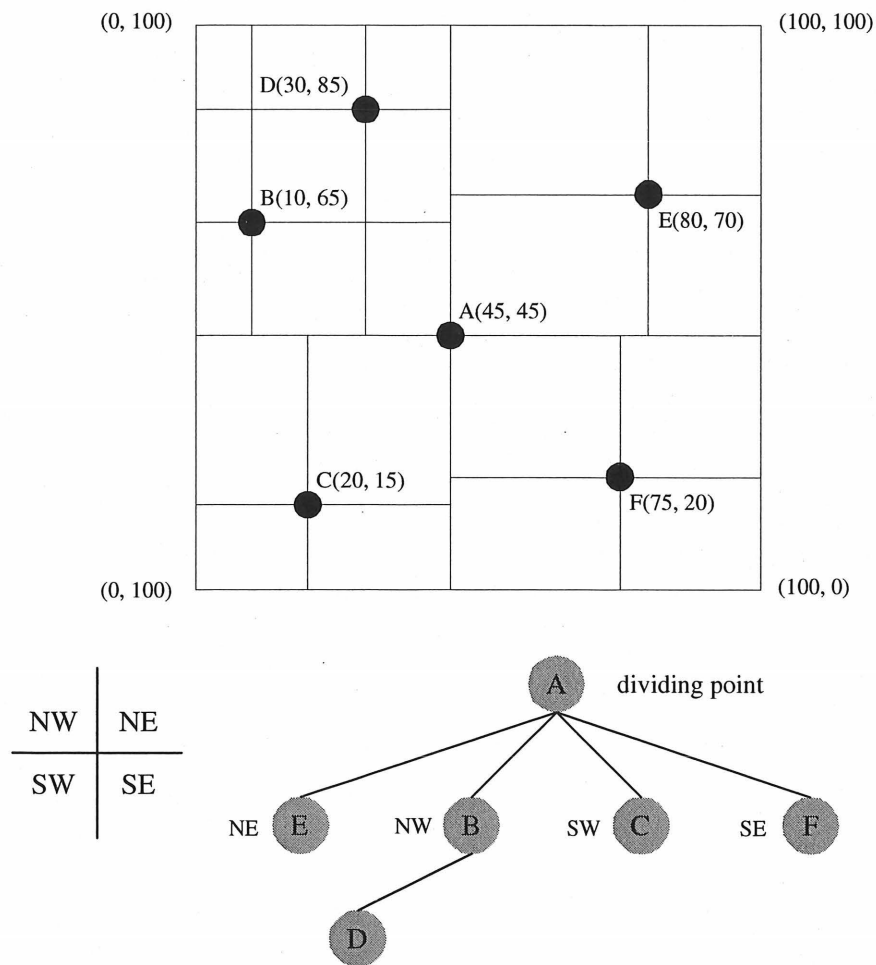


Figure 2.4: A two dimensional point quad-tree

If in a region quad-tree we consider each black pixel as representation of a point in the low left corner of the corresponding pixel it is a MX quad-tree. As in region quad-tree the resolution is as much as the smallest pixel. If in a region quad-tree we continue the

splitting until each quadrant (pixel) contains only one point it is a PR quad-tree. In a PR quad-tree we store the coordinates (spatial location) of the point objects as part of the object data so we have the exact spatial coordination of each point object.

**k-d tree** This structure is suitable for range queries. In a k-d (k dimensional) tree the dimension of the split is variable and for each node can be different. In each splitting node that  $j^{th}$  attribute ( $p_j$ ) is the discriminator all the points with their  $j^{th}$  attribute smaller than  $p_j$  will be on the left (low) son (LOSON(p)). The points with their  $j^{th}$  attribute bigger than  $p_j$  will be on the right (high) son (HISON(P)) (see figure 2.5).

There are many modified versions of k-d tree. For example k-d-B tree is a modified k-d tree with paging capability. It is useful when the indexed data is too big to be put completely in memory.

We can name other famous point indexing structures as: Bv tree, HB tree, projection method, grid file and Excell. The list is growing.

## 2.4.5 Rectangle indexing structures

To index a non zero dimensional object we can approximate it with a Minimum size Bounding Rectangle (MBR). Handling the MBR that represents the non zero dimensional object is much easier than handling the object itself and have only 2k attributes in a k dimensional space. There are many indexing structures to index these rectangles. We can name some of them as: PLOP Hashing [KRIEGEL, 1988], quad-CIF-tree [FUSSELL, 1981, SAMET, 1984], locational keys [ABEL, 1983]. Matsuyama's k-d tree [MATSUYAMA, 1984], Cell tree, Buddy tree, corner stitching, R-tree [GUTTMAN, 1984],  $R^+$ -tree,  $R^*$ -tree and other versions of R-tree. The most famous ones are R-tree and its different versions. Here we have a short look at the basic R-tree,  $R^+$ -tree and  $R^*$ -tree.

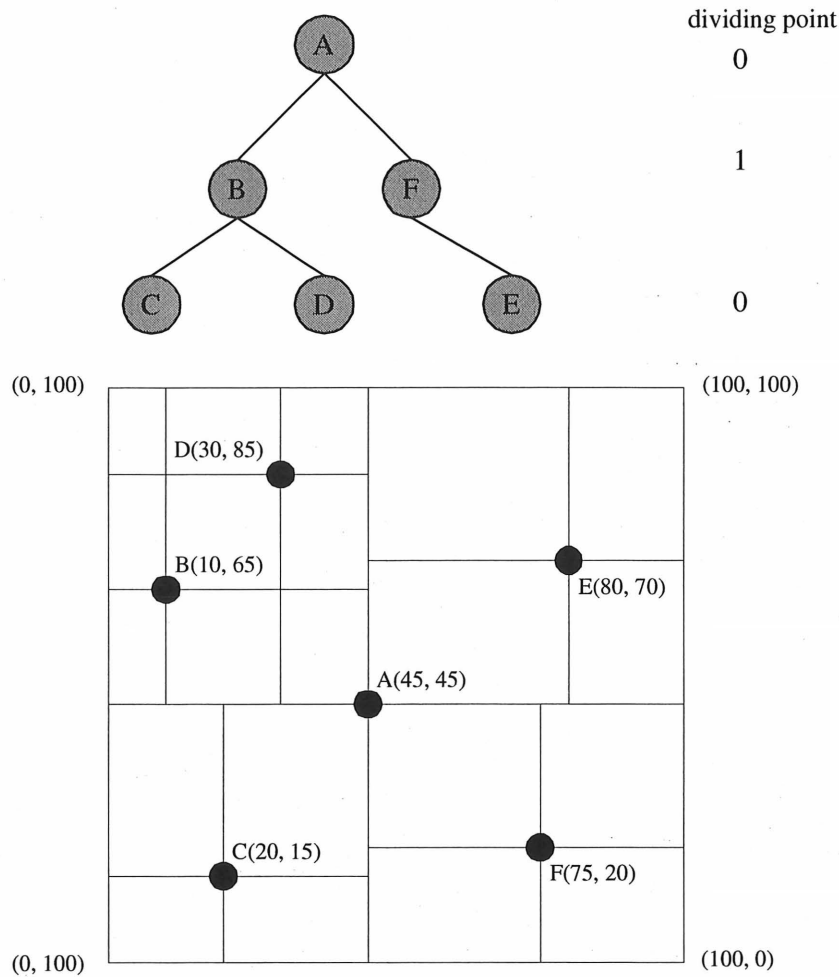


Figure 2.5: A two dimensional k-d tree

**R-tree** R-tree is a multidimensional generalization of B-tree. Each leaf node may contain many objects of the form:  $(I, \text{object-identifier})$  in which  $I$  is the MBR of the object and *object-identifier* refers to the object itself. Non-leaf nodes contain entries of the form:  $(I, \text{child-pointer})$  in which  $I$  is a rectangle in a lower level of R-tree that contains all the objects contained in that sub tree. As shown in figure 2.6, in R-tree, rectangles can overlap. This is a disadvantage of R-tree during a search. Depending on the query range we may need to search more than one overlapping rectangle.

To insert a new object in the tree the rectangle that needs minimum enlargement is selected. If there are more than one then the one with the smallest area. There is a level of overflow that if the insertion reaches that level a split must occur. The split

may propagate to higher levels of the tree. The same thing can happen for a delete. If the number of objects in a rectangle decrease to less than a minimum level the node is deleted and its contents is reinserted to the tree. A delete may propagate to upper levels.

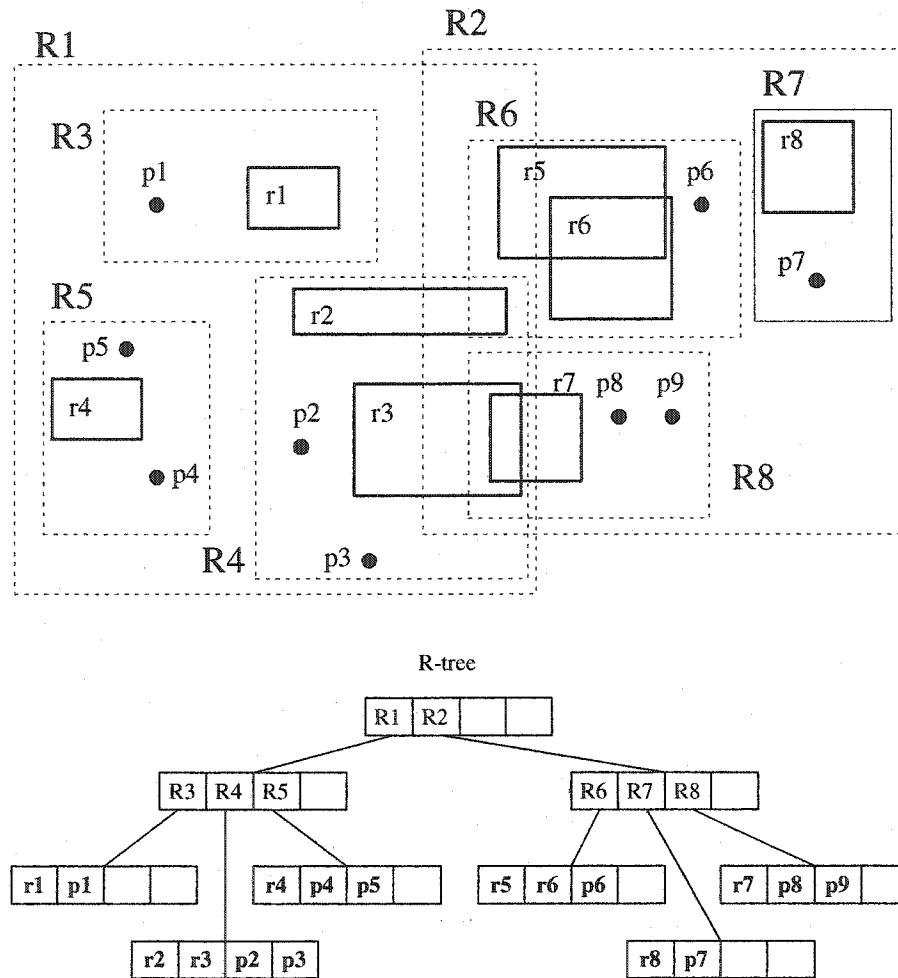


Figure 2.6: The structure and planar representation of R-tree

**$R^+$ -tree**  $R^+$ -tree proposed to solve the problem of overlapping rectangles of R-tree. As shown in figure 2.7 every common object is repeated in the sharing rectangles and the rectangles remain disjoint. This is an advantage for search queries but it also creates some complications as dead space problem. The dead space problem results from

the enlargement of rectangles are in contradiction to each other and a region remains uncovered by neighboring rectangles. Many strategies are proposed to solve the problem.

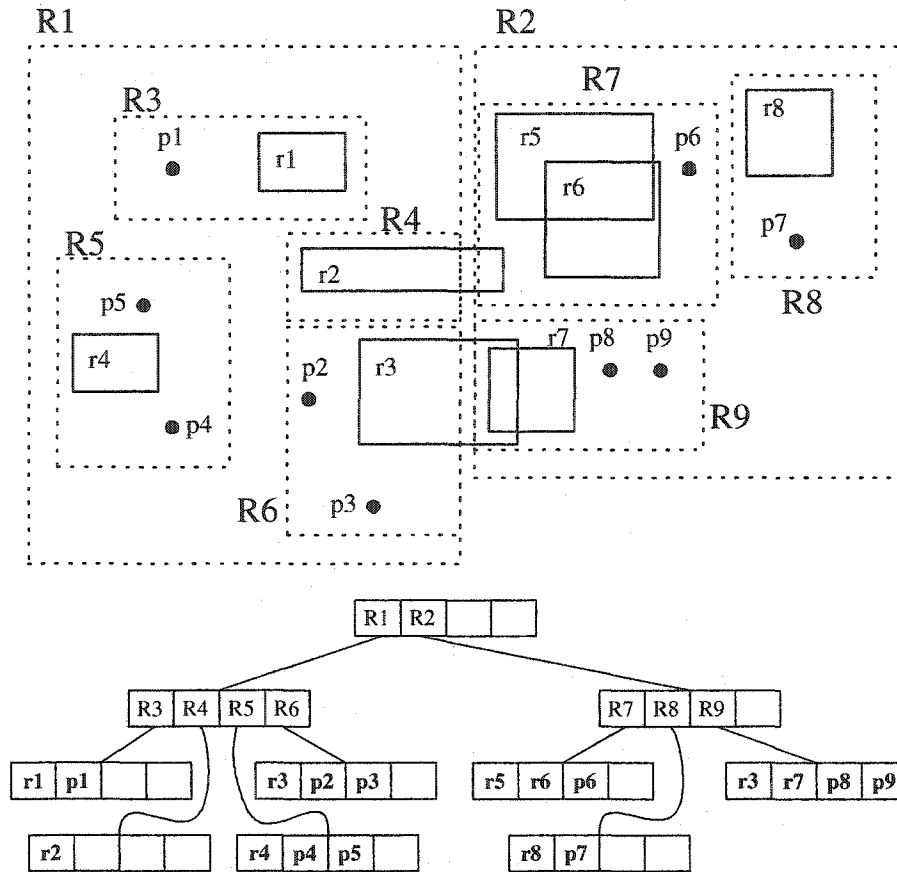


Figure 2.7: The structure and planar representation of  $R^+$ -tree

**$R^*$ -tree**  $R^*$ -tree [BECKMANN, 1990, THERIAULT, 1996] is an organized version of  $R$ -tree. In  $R$ -tree the structure of the tree depends on the order of insertion and deletion of objects rather than the objects themselves. In  $R^*$ -tree the structure of the tree is reorganized after each insertion or deletion by applying a set of rules.

## 2.4.6 A general view of spatial access methods

In figure 2.8 we summarized all spatial indexing methods. The methods are categorized in two categories based on the type of the objects they index, zero dimensional objects and non-zero dimensional objects. We can see all methods for indexing zero dimensional objects can be applied for indexing non-zero dimensional objects by using the following concepts:

1. Rectangles or other polygons can be considered as points in a higher dimensional space. For example a  $k$  dimensional rectangle can be seen as a point in a  $2k$  dimensional space.
2. Complicated objects and regions can be approximated by MBR or other minimum bonding polygons.

Application of these two concepts gives a flexibility in choosing the most efficient method for indexing spatial objects in a database.

## 2.4.7 Graph modeling and indexing

For some spatial applications the foregoing spatial indexing methods are not enough to handle the queries. In road map databases we need a graph indexing and a graph traversal approach to evaluate the queries [ZHAO, 1994]. Graph indexing and traversal approaches are useful for a wide range of applications such as congestion management, travelling information systems, transportation and dispatching. All GIS products that contain the maps of cities, maps of electrical networks, communication networks, gas lines are examples of application of graph indexing structures. Also one application that recently has become very popular is the GPS (Global Positioning System) systems in cars that help the driver drive to a specific address in a city. In these databases

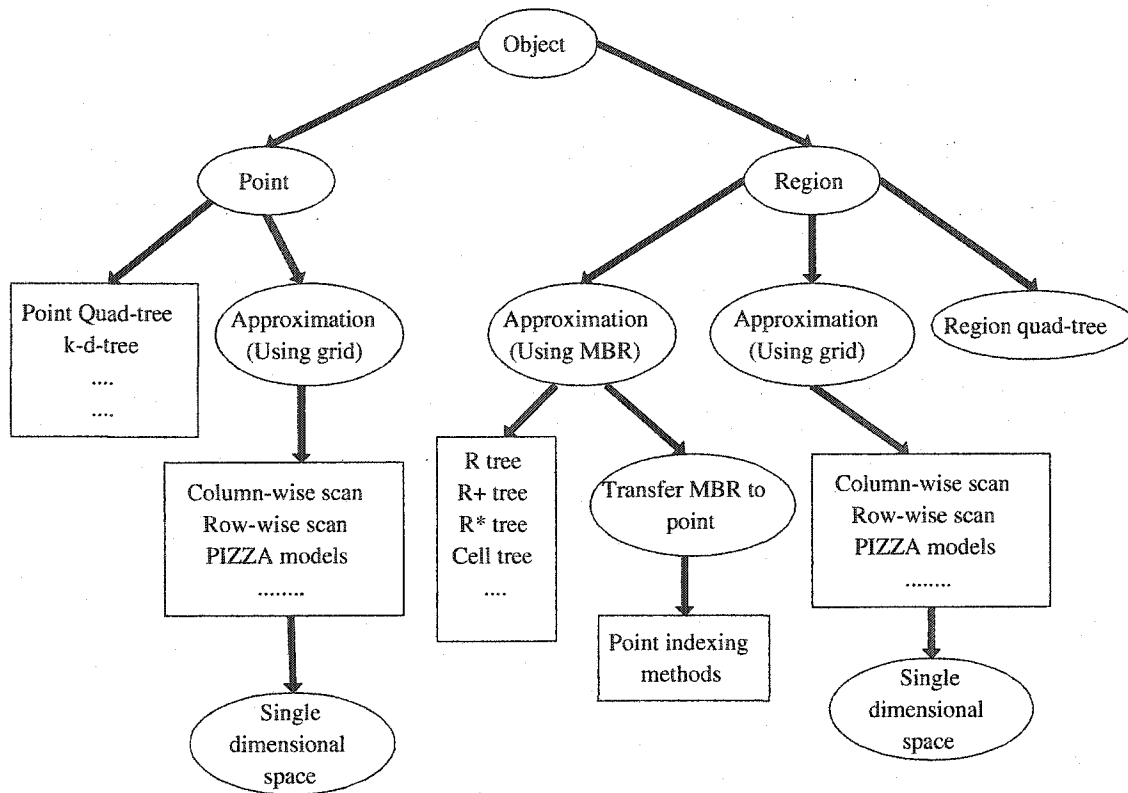


Figure 2.8: Spatial indexing methods

the information is stored in a topological form of nodes and links and the queries can be spatial (Point and range queries, neighborhood, intersect, etc.) or graph traversal queries.

In section 2.1 we brought a set of example queries based on a road map database. The examples contain both spatial and graph traversal queries. Interpreting the queries in Example 1 and 5 leads to a neighborhood query, the query in Example 2 leads to a range query that will be answered and the queries in Example 3 and 4 leads to an intersect query. The spatial indexing technique that we brought in previous sections are enough to evaluate the queries in all examples 1 to 5. R-trees can be used for neighborhood queries, k-d trees for range queries and in all of them MBRs can be used to approximate the objects or to prune the search space during the filtering phase of query processing.

Interpreting Example 6 of section 2.1 leads to a different kind of query, a graph traversal query. Evaluating a graph traversal query needs graph indexing structures and methods.

#### 2.4.8 Evaluation of the spatial indexing structures

Indexing structures for spatial database systems must keep the information about spatial order and spatial positions of the objects in the most efficient way. To evaluate and compare the efficiency of the indexing techniques we must evaluate the following parameters. As we will see while an indexing technique is efficient for an application it may be inefficient for other applications. The evaluation parameters are:

1. Required resource (memory or secondary storage) to hold the indexing structure
2. Required time to construct the indexing structure from raw data (non-indexed data)
3. Required time to update index structures: whenever an insert or a delete operation is done on the database objects, the indexing structure must be updated. This may take a considerable time. In some indexing techniques the update time for a delete operation is quite different from the update time of an insert operation.
4. Supported queries: As we said there are different kinds of basic spatial queries like point queries, window queries and neighborhood (nearest and farthest) queries. Each indexing technique may support some of them.
5. Search efficiency: The time to perform a basic query by using the index structure. This time is different for different kinds of queries.
6. Reliability (redundancy and recoverability): The redundancy in indexing structure determines the ability to recover errors in the indexing structures, which is effective on the reliability of the system.

The importance of these parameters depends on the application. All indexing techniques are modified versions of a few principal indexing structures. For example to improve the search time by eliminating the problem of overlapping MBRs in  $R$ -tree, the  $R^+$ -tree was introduced. But in return it has a problem with inserting new objects in the index structure. In fact different sequences of insertion of the same objects creates different trees. To solve this problem  $R^*$ -tree was introduced. Another example is k-d-B-tree that is a modified version of k-d-trees to improve its paging capability for efficient secondary storage (e.g. hard disk). This is useful when the size of the index is too big to be kept in memory and must be stored on the secondary storage device.

More information on different versions of spatial indexing structures can be found in [SAMET, 1991a, SAMET, 1991b].

## 2.5 Spatial query optimization

In spatial DBMS like conventional DBMS efficient query processing depends on indexing techniques and optimization strategy. we presented some basic concepts about indexing techniques in previous section. Here we present basic concepts of query optimization. A more complete survey of the subject for advanced databases can be found in [FREYTAG, 1994].

Query optimization is the process of finding an efficient strategy for executing a query. In low level languages the efficiency of information retrieving is highly dependent on the programmer's skill but as the level of language goes higher the efficiency goes out of the hands of programmer and the importance of optimized interpretation increases. This is the reason for the efforts that is still being done to find better optimization strategies in new DBMS with high level user interfaces. The optimization is usually done according to the time and cost of query processing. The optimization process imposes an overhead

to the system that takes resources. The overhead imposed by running the optimizer itself, must not exceed the improvements of cost and time of query processing achieved by optimization.

The query optimization for all kinds of databases is done through several steps. In each step, depending on the application and the database a set of concepts and techniques are used. Figure 2.9 shows the flowchart of a typical query optimizer. The figure summarizes the query optimization procedure in two main steps as following:

- Query transformation
- Application of a set of optimization techniques

Other blocks (Cost Mode, Algebraic Space and Method-Structure Space) are used by these two main steps to estimate the cost of primitive operations, to apply transformation rules and to process the primitive operations.

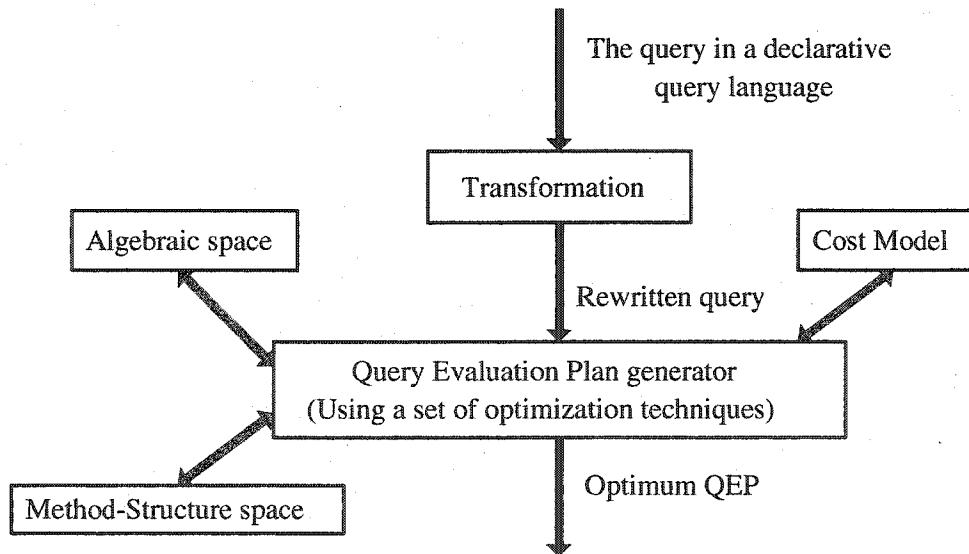


Figure 2.9: Basic structure of a query optimizer

### 2.5.1 Query transformation

The most important step of query processing in every type of databases is query transformation [JARKE, 1989a]. In this step the query is rewritten through a set of algebraic operations to achieve a more efficient and simplified form of the query. The complete transformation approach consists of the following three steps. In simple query processing approaches only one or two of them can be included.

1. Standardization
2. Simplification
3. Amelioration

In the standardization phase the queries are modified from a syntax point of view. The starting point is to constitute parse tree as an internal presentation of the query. The logical part of the parse tree that represents the constraints and predicates is used for further steps. In the next step by use of logic rules (De Morgan, Associative, Commutative and Distributive) the query is changed to one of the two standard forms: CNF (Conjunctive Normal Form or product of sums) or DNF (Disjunctive Normal Form or sum of products). Each form has its advantages.

During simplification phase, the simplest equivalent of the query is achieved, based on the redundancy of the query and the use of the simplification rules. The result of simplification phase is not necessarily the most efficient one.

During the amelioration phase, the semantic of query is used to achieve more efficient equivalent of the query. For example in relational databases, replacing successive projections by one equivalent projection can be done in this step. Replacing negative predicates by their positive equivalent is also done in this step.

### 2.5.2 A set of optimization techniques

There are a set of optimization techniques that may be used to achieve a more efficient equivalent of the query. The order of application of these techniques is not specific and determined: how and where to use these methods depend on the designer and the database.

**Semantic optimization** Based on the semantic analysis of a query we can replace subqueries with their semantically equal expressions, eliminate redundant subqueries or add new subqueries based on the integrating rules of the database to rewrite a query in simpler and more efficient configuration [ABERER, 1994, NIGAM, 1997, BELL, 1997, FRIAS, 1996, LEVY, 1995].

**Loop optimization** One important optimization technique used by compilers is loop optimization. The same technique can be used for query processing in databases. With this optimization step the contents of loops is pulled out as much as possible and the equivalent non-loop operation is added to the query. Depending on the query, sometimes all of the loop can be replaced with an equivalent non-looped query. This loop simplification or elimination decreases the query execution time at the cost of additional memory usage. Indeed loop optimization is a time optimization technique.

**Join optimization** In relational databases many methods for join calculation has been developed. Depending on the application characteristics the most efficient method is chosen from one of three main methods: nested-loop method, sort-merge method and hash-based method. For multiprocessor systems we have other methods. More detailed information can be found in [VALDURIEZ, 1984, GALINDO, 1997]. In spatial databases the computation of spatial joins is completely different from conventional

DBMS yet they are comparable in the sense that both in spatial join and in conventional join every Object/Tuple of the first Relation/Map is involved with all Objects/Tuples of the second Relation/Map.

The basic operations in a spatial database are point query, window query and spatial join. The last one is the most time consuming one. Depending on the spatial indexing structure of the database the spatial join queries can be implemented in different ways. Many join calculation and other algorithms dealing with R-tree based spatial structures were introduced in [MARTYNOV, 1996].

One basic technique that is used in spatial join calculation is the filtering and refinement technique that is implemented in two steps. In the filtering step all MBR of each joining spatial relation that intersect a MBR of another joining spatial relation are detected. In the next step (refining), all the objects in intersected MBR will be checked for intersection and the objects that intersect each other from different spatial relations will be determined.

**Early restriction** The number of the objects that a query is performed on is a determinant factor in the query processing time. For queries that are combined from a set of predicates (subqueries), predicates that restrict the field of search more than the others, must be evaluated first. The relevant parameters in choosing the order of performing predicates or subqueries, are the restriction that each query puts on the field of search and the cost of evaluation of the subquery.

**Spatial reasoning** Spatial reasoning can be used in semantic optimization to find the contradictions or redundancies in a query and to rewrite the query in a simpler semantically equivalent form. Also in a deductive spatial database (e.g. see [LU et al., 1995]), spatial relationships are specified by deduction rules. The deduction rules are used in combination with the spatial reasoning for query and optimization.

**Physical access optimization** In the final step of processing a query, the query is broken down into the lowest level operations such as basic storage and other peripheral device access operations. These low level data access operations can be managed and optimized based on the number of users, physical storage device characteristics and the amount of data access request. The optimization technique and its importance differs from system to system. As an example in Memory-Resident databases the issue differs significantly from that in conventional disk resident databases [WANG, 1990]. For concurrent, networked and multiuser databases, this kind of optimization has a greater effect on the overall efficiency of the system.

### 2.5.3 A generic model for a query optimizer

As we discussed the subject of query optimization contains a wide range of concepts and methods. Here we present a generic model for a query optimizer in database systems that can sum up all these techniques in a basic model and give a clearer view of the subject.

The process of query optimization is shown in figure 2.10. This process contains operations and procedures in three classes:

1. Rewriting and equivalent generation
2. Breaking down to lower level
3. Cost evaluation and search for the most efficient plan

Through the process of query optimization, the query passes different levels of interpretation. The highest (first) level is the user interface level or language level (the query in the query language text). Next level is the algebraic expression of the query. The last level is a QEP which consists of basic storage and peripheral device access operations.

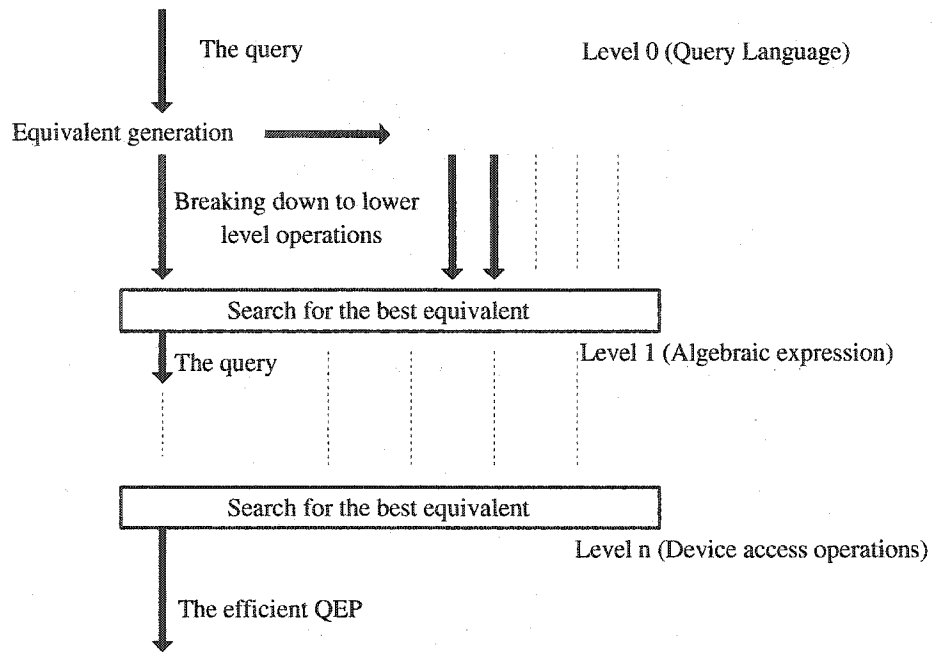


Figure 2.10: Generic model for a query optimizer in database systems

For each level of query we may have (but not necessarily) these three classes of operations: rewriting, search for the most efficient plan and breaking down to lower level as in the figure 2.10. All concepts and techniques that are discussed under the subject of query optimization can be seen as a part of this model. For example join optimization in the above model is a combination of second and third class (Breaking down to lower level and Search for the most efficient plan) in algebraic level. Semantic optimization can be seen as a combination of first and third class (Rewriting and Search for the most efficient plan) in algebraic level. Loop optimization can be seen as a combination of first and third class (Rewriting and Search for the most efficient plan) in first level (query language text).

## 2.6 Spatial query processing architectures

There are many concepts that can be used in the architecture of a spatial query processing system. Here we will have a short look on some of them.

### 2.6.1 Separating spatial from aspatial queries

Spatial databases can be considered as a combination of spatial and aspatial data. If we separate successfully the queries into these two categories we can use the conventional techniques for the aspatial part and the new techniques for the spatial part. According to the different systems that have been introduced we can generally show the query processing procedure as in figure 2.11. An early work of Ooi [OOI, 1991] includes a discussion of this approach.

First, a primary optimization is done and the result goes to the decomposer to separate the spatial and aspatial parts of the query. The another level of optimization can be done on the aspatial part that is treated as conventional query. The spatial part goes to the spatial optimization step that we explain later. The results of the two branches are then put together by use of a sequencer and then final optimization (physical access optimization) is done. Finally the Query Evaluation Plan that is generated will be executed and the results of the query will be achieved.

Separating the queries into spatial and aspatial gives us the possibility of using existing conventional database for the aspatial part but it doesn't necessarily lead to the best and most efficient query evaluation plans.

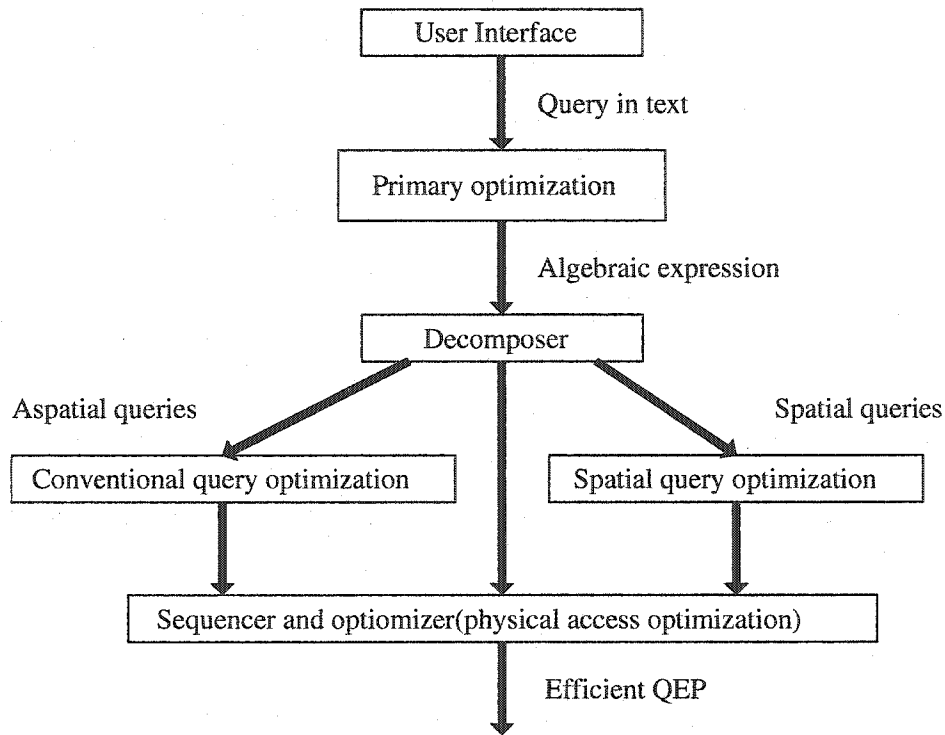


Figure 2.11: An example architecture of a query processor in spatial databases

### 2.6.2 Spatial filtering

An efficient approach to interpret the spatial part of queries is the filtering and refinement procedure. In most spatial systems, the spatial part of queries is interpreted by a two phase procedure:

1. Filtering phase
2. Refinement phase

During the filtering phase, MBR of objects are used to prune the search space. This way, the objects whose MBR don't fit the query predicates are eliminated. Next, during the refinement phase, a smaller set of objects remaining from filtering phase is processed. An early work using this concept is presented in [HWANG, 1994]. A more advanced optimization approach based on this concept is presented in [HO-HYUN et al., 1999].

### 2.6.3 Extensible architectures

Although each database has its own specification, the process of query optimization uses general concepts for every kind of database. The idea of having an extensible query optimizer that can be casted according to the specification of the database was first realized by the EXODUS optimizer generator [GRAEFE, 1987] and then completed by the Volcano optimizer generator [MCKENNA, 1993] and the open OODB query optimizer introduced in [BLAKELEY, 1993].

In these systems the query optimizer is generated based on the specifications of the database. These specifications are:

1. Algebraic operator
2. The set of algebraic transformation rules
3. Statistics and cost models
4. Physical data access structures and algorithms.
5. Search strategies

Figure 2.12 shows the optimizer generator paradigm. The inputs to the process (query optimizer generator) are:

1. Model file (Supplied by the implementor)
2. Implementor supplied functions (e.g. cost functions)
3. Model-Independent code(e.g. search engine)

and the output is a query optimizer for the database that whose specification is given in model file. The model file contains the operators declaration, logical transformation rules

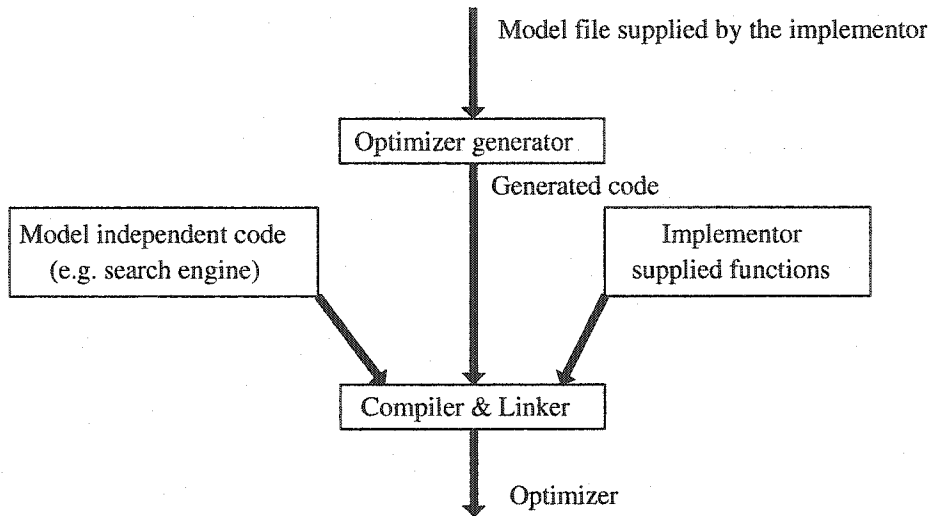


Figure 2.12: Optimizer generator paradigm

and implementation rules (Rules that map operators to their implementing algorithms). The code of the query optimizer for the specified database is generated based on this model file. However, this code is not the complete code of the query optimizer. There is some model-independent code (e.g. search engine) and implementor-supplied code (e.g. cost functions) that must also be compiled and linked together to achieve the query optimizer.

The quality of the generated query optimizer depends on:

1. The transformation rule set: If it isn't a complete rule set (if it doesn't generate all possible equivalent expressions of the query) the optimal evaluation plan might not be achieved by the generated query optimizer.
2. The implementation algorithms
3. Search engine and other model independent codes.

For spatial database systems, the issues of implementation algorithms and transformation rules are more serious than that of the other kind database systems. Because of the

different nature of primitive spatial operations that can be run in two phases of filtering and refinement, we will have transformation algorithms rather than transformation rules. Using an extensible architecture (as shown in Figure 2.12) for spatial databases may help to implement a query optimizer easier but not necessarily more perfect than the one designed from scratch [BECKER, 1992]. The extensible paradigm is only for the application of transformation rules and not the application of transformation algorithms.

## Chapter 3

# A BRIEF AND GENERAL LOOK AT THE WORK

This chapter will have a brief and general look into what we accomplished during this work. Details of the work will come in next chapters.

This work is a solution to the problem of efficient query processing in spatial databases. To that end, existing query processing techniques and concepts in different databases were analyzed. The current spatial systems and applications were investigated, as in chapter 2. Then a new algebra and new query optimization techniques were introduced.

### 3.1 A new map algebra

While other works have used a modification of conventional and existing approaches such as deductive, relational, and object-oriented models to handle spatial data, here a new map algebra is introduced. The algebra is presented thoroughly in chapter 4. Data elements, operators, and theorems are defined and proved.

There are two reasons that an algebra for spatial databases must be different from the

algebras for aspatial databases (Relational, Object-oriented, Object-relational, etc.):

1. Even if we use appropriate indexing structures the query processing time in spatial databases is proportional to the number of objects involved (having two steps: filtering and refine, the filtering step can be logarithmic while refinement step is always proportional) but in aspatial databases the processing time is a logarithmic function of number of objects or tuples.
2. The set of operators for spatial data are totally different in behavior to the set of operators on aspatial (e.g. compare spatial join and relational join).

The new algebra is invented to address the above differences. The algebra will be used for applications with very large spatial databases which can be organized in as many separate maps as possible. As in other models, the new map algebra is a descriptive algebra rather than a prescriptive, so it is used only for query and not for editing.

## **3.2 New query optimization and processing techniques**

Figure 3.1 shows the application of this work in a spatial DBMS. The query optimizer receives a query in input and produces a set of primitive operations. The evaluation of this set of primitive operations will produce the query result. Efficient processing of QEP depends on the indexing structures but before that, efficient interpretation of the query to QEP depends on the query optimization techniques. In this work some new optimization techniques are introduced using the new map algebra. Chapter 5 explains the query processing steps and query optimization techniques of this work.

The new introduced optimization technique are:

1. Multi-Level filter/refinement that is an improvement to the existing filter/refinement approach.
2. A QEG generation and transformation method. Transformation rules are introduced and discussed.
3. Predicate ordering: It is shown that the order of evaluating combinational predicates is determining in the processing cost. A method is presented to find the best equivalent of the combinational predicate.

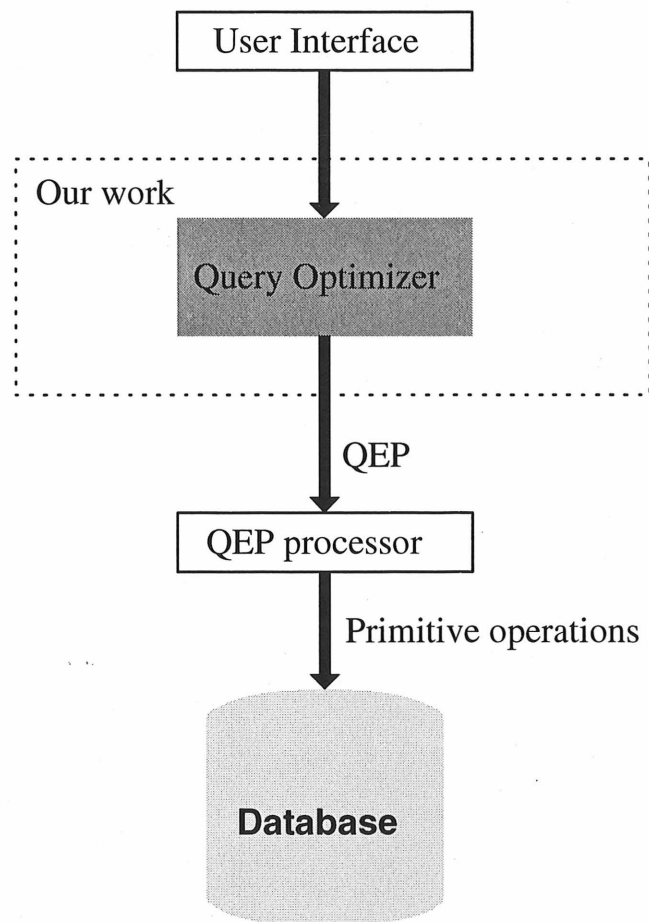


Figure 3.1: Our optimizer in a spatial DBMS

Although only the new concepts and techniques are used in the implemented system but it is possible to use this work in combination with other approaches as shown in Figure 3.2. As far as these approaches produce an algebraic expression of the query (rewrite

the query in a more efficient expression) they can come before our query optimizer. This doesn't reduce the importance of our system because the main challenge is how to break the query down to primitive operations in an efficient way rather than rewriting it.

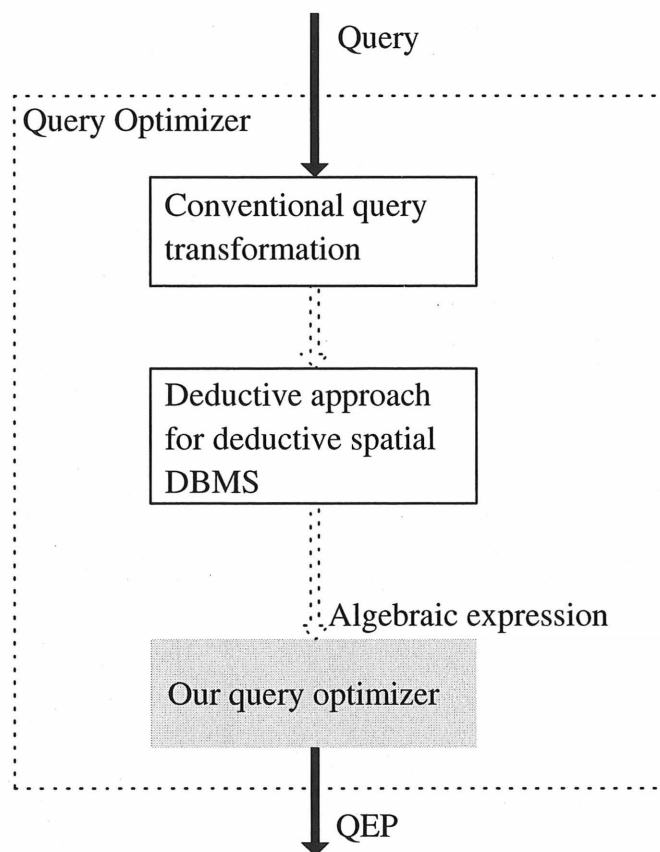


Figure 3.2: Potential use of our work in combination with other approaches

### 3.3 The implemented system and experimental results

To put the new concepts into experiment a query processing system was designed and implemented in a simulated environment. Then randomly generated queries were put to the system to get the estimated cost before and after application of the optimization

techniques. Chapter 6 explains the implemented system and the experiments. This proved two points:

1. The complete query processing procedures and algorithm we introduced works correctly.
2. The application of the new optimization techniques improves the cost (processing time and allocated resources) of QEP evaluation remarkably. The bigger portion of the improvement is achieved by the application of the new multi-level filter/refinement approach.

## Chapter 4

# THE SPATIAL FORMALISM AND ALGEBRA

In this chapter we propose an object-based formalism and algebra for spatial systems. The goal is to write queries in a topological and metric space and later express query optimization procedure and rules. The algebra looks at the spatial data as maps (classes) of objects. There are operations for both maps and objects, so we will have a multi-set algebra. The taxonomy may look like object oriented paradigm but it is in fact a different one. There is no hierarchy or inheritance among maps. In fact maps are more similar to relations rather than classes. The same universal coordinates are assumed in all maps of a database. The coordination of an object in a map gives its coordination in all other maps of the same database. We don't define boundaries for a map because practically the boundaries of a map is as far as its objects. As you will see through next chapters, the application of this model is for optimized query processing in spatial databases with very large number of objects represented in different maps with one universal coordination. To have a better understanding of the definitions and concepts, we use examples through the chapter. At the end some transformation rules (useful in query optimization) are introduced.

## 4.1 Data elements

Consider a GIS database containing the maps of a country with cities, lakes, airports, etc. Figure 4.1 shows three maps of this database, a map for each set of objects. The three maps could be represented in one map as in Figure 4.1-d but for query processing efficiency reasons, in our model a spatial database should be designed and presented in many maps depending on the nature of spatial data and the application (queries). Each object in the maps contain spatial and aspatial information of its corresponding Real World Object (RWO). Although we deal mostly with spatial properties, we will have some solutions and suggestions for aspatial part of objects as well. The algebra is able to express and process queries based on both aspatial and spatial part. To accomplish this objective we have defined several data elements.

A database consists of objects and maps of objects, defined as follows:

- Object: An object consists of three attributes:
  1. MNM Map-name, the name of the map the object belongs to
  2. APT Aspatial-part, representing all alphanumeric data of the object
  3. SPT Spatial-part, representing the geometry of the object

An object is represented by its OID (Object Identifier), a unique number for each individual object.

- Map: A map consists of a set of objects that have a spatial relation together. In practice a map represents a spatial index (Although the algebra is unaware of spatial indexing). Maps are different to relations because a relational database (either first normal form or 2nd or third) is designed based on the relationship among alphanumeric fields of tuples. While maps should be defined (designed)

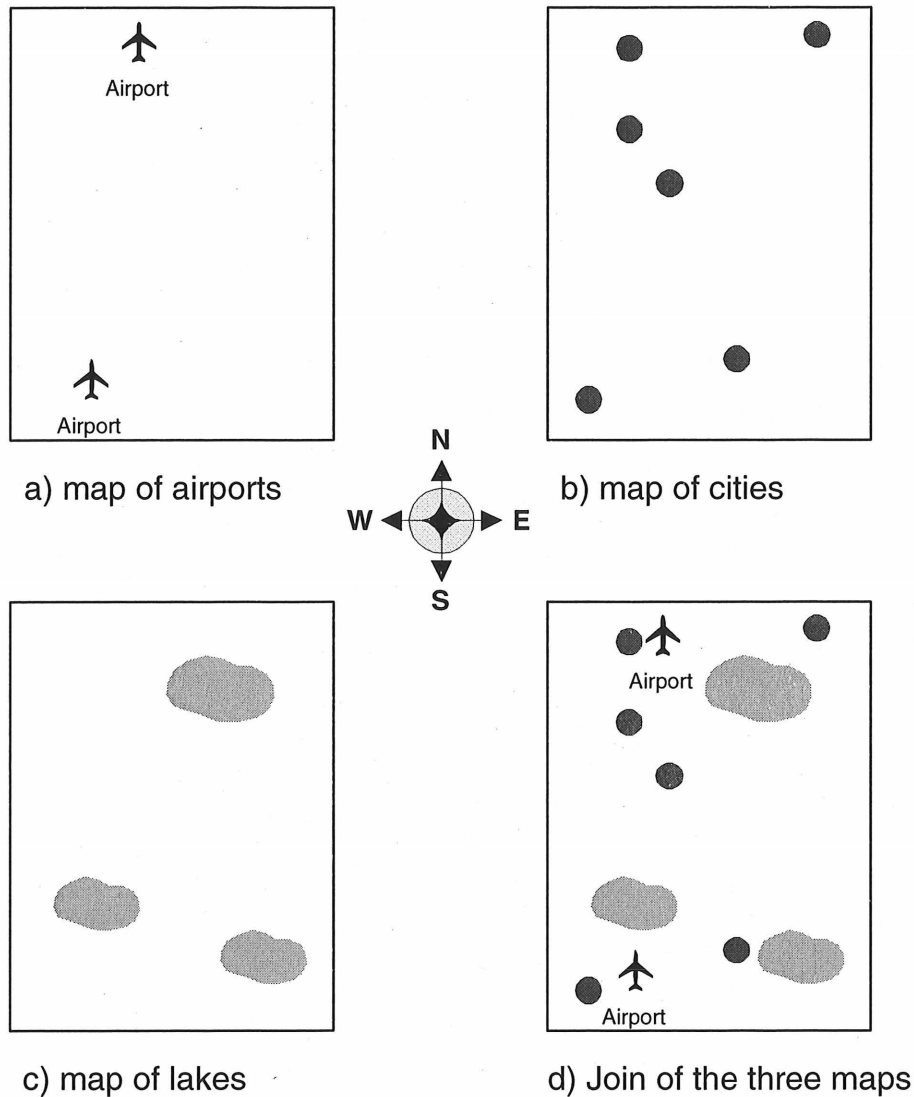


Figure 4.1: A spatial database

based on the spatial relationship and spatial information of the objects. So, it won't necessarily be efficient to define a map equivalent of each relation to handle spatial data in a relational database. However as a provision to support the existing and new query languages and databases we may look to the maps as Relations or Classes. One may look to the maps as Classes of Objects when supporting Object Oriented query languages (e.g. SQL:99). At the same time maps can be regarded as Relations when supporting Relational query languages.

In algebraic expressions, a map is represented by its name.

To express queries on the database we use other data types and definitions as follows:

- **Coordinates:** a vector of  $n$  scalars in a  $n$  dimensional space. It can represent a vector or a point.
- **Window:** a window in a  $n$  dimensional space is defined by its  $n$  points (Coordinates).

## 4.2 Map operators

Maps are sets of objects (Comparing to relations in relational model that are sets of tuples), having almost similar operators to set operators. The operators we introduce are not necessarily primitive and some of them can be defined in terms of the others. They have closure property (Operations on map(s) always result map(s)). We represent them in two categories: monadic and dyadic. A monadic operator has one operand. A dyadic operator has two operands.

**Monadic map operations** Here we introduce Selection, Map-Window, and Map-Point operators. Map-Window and Map-Point can be seen as special cases of Selection. But distinguishing them from Selection helps the optimizer to produce a more efficient QEP.

- **Selection:** receives  $mp$  of type of Map and an object predicate (let  $prd$ ) as inputs. The result is a map containing all objects in  $mp$  that meet the predicate  $prd$ . The algebraic expression of the operator is as follows:

$$\sigma(mp, prd) : \{x \mid x \in mp \wedge prd(x)\} \quad (4.2.1)$$

- **Map-Window:** receives a map (let  $mp$ ) and a window (let  $wn$ ) as inputs. The result is a map containing all objects in  $mp$  with their gravity-center in  $wn$ . The algebraic expression of the operator is as follows:

$$\varpi(mp, wn) : \{x \mid x \in mp \wedge x \in wn\} \quad (4.2.2)$$

- **Map-Point:** receives a map (let  $mp$ ) and a point (let  $pnt$  of type Coordinates) as inputs. The result is a set of objects that cover the point. The algebraic expression of the operator is as follows:

$$\pi(mp, pnt) : \{x \mid x \in mp \wedge pnt \in x\} \quad (4.2.3)$$

**Dyadic map operations** Here we introduce Join and set operators. They receive two maps as input and produce a map as output. Although the set operators are a special case of the join operator, we define them separately in the algebra. This simplifies the query processor and optimizer, making it easier to distinguish these special cases of join and interpret them differently.

- **Join:** receives two maps (let  $mp_1$  and  $mp_2$ ), an optional predicate (let  $prd$ ) and output specifier (let  $out$ ) as inputs. The result is a map containing objects in one or both maps that meet the predicate ( $prd$ ). The algebraic expression of the operator is as follows:

$$\bowtie (prd, out, mp_1, mp_2) : \{out(x) \mid x_1 \in mp_1 \wedge x_2 \in mp_2 \wedge prd(x_1, x_2)\} \quad (4.2.4)$$

where:

$$out = [x_1 \mid x_2 \mid x_1 + x_2]$$

- **Set operators:** There are three basic set operators that receive two maps as sets of objects in input. The output map is a set of objects that is obtained by adding, subtracting or intersecting the input sets. The algebraic expression of the operators are as follows:

1. Union:

$$mp_1 \cup mp_2 : \{x \mid x \in mp_1 \vee x \in mp_2\} \quad (4.2.5)$$

2. Difference:

$$mp_1 \Delta mp_2 : \{x \mid x \in mp_1 \wedge x \notin mp_2\} \quad (4.2.6)$$

3. Intersect:

$$mp_1 \cap mp_2 : \{x \mid x \in mp_1 \wedge x \in mp_2\} \quad (4.2.7)$$

Table 4.1 summarizes the introduced map operators.

## Definitions

- The object variable representing objects of a map is  $\lambda$  followed by the name of the map. For example the object variable of the map named *City* is  $\lambda City$ . The object variables used in the predicate of a map operator can be presented as  $\lambda_i$ . Where  $i$  is an index to the corresponding map.
- The aspatial part of an object is treated as an object of an aspatial database. The object variable representing aspatial part of objects for a map is  $\lambda$  followed by the map name followed by *.APT*. For example the object variable representing aspatial part of objects for a map named *Building* is  $\lambda Building.APT$ .

## 4.3 Object operators

As we mentioned in the previous section, there are two sets of operators: map operators and object operators. The object operators receive one or more objects in input and give an object or a map in output. Object operators are used to express predicates in map operators.

**Object-editing operators** This group of operators receives one or more objects as input and produces one of type coordinate as output. The output object attributes can be a function of both spatial and aspatial attributes of input objects. Here we present a few example operators.

- **rotate**: Receives an object and a parameter of type coordinate in input. The object in the output will be a rotation of the input object according to the input parameter. Similar to rotation we can define an operator to simulate translation. The algebraic expression of the operator is as follows:

$$y = rotate(x, d):$$

- $y.APT = x.APT$
- $y.SPT = \text{rotation of } x.SPT \text{ by } d$
- $y.MNM = x.MNM$

- **intersect**: Receives two objects in input. The output is an object whose spatial part is the intersection of the two objects with empty aspatial part. Similar operators can be defined for adding or subtracting the aspatial parts of the two objects. The algebraic expression of the operators is as follows:

$$y = intersect(x_1, x_2):$$

- $y.APT = \emptyset$
- $y.SPT = \text{intersection of } x_1.SPT \text{ and } x_2.SPT$
- $y.MNM = x_1.MNM = x_2.MNM$

**Conversion to map** To give the possibility of creating new maps from single objects we define the following operators:

- **convert:** Receives one object as input and produces a map as output. In other words it creates a map from a single object. The algebraic expression of the operator is as follows:

$$\text{convert}(x) = \{x\}$$

- **insert:** Receives one map (let  $mp$ ) and one object (let  $obj$ ) as input and produces a map as output created by adding the input object to the input map. The algebraic expression of the operator is as follows:

$$\text{insert}(mp, obj) = \{x \mid x \in mp \vee x = obj\}$$

## 4.4 Aggregate functions

Aggregate functions are functions that receive a map or object as input and return a value of the atomic defined types (APT, SPT, window, coordination, etc.). Depending on the application, we can define our own aggregate functions. For example functions to calculate volume or perimeter of the spatial objects. Here we bring some common examples of aggregate functions.

- *gravity – center*( $x$ ): Calculates the central gravity point of the input object. The result will be of type coordinate.

<i>Map operators</i>	
Operator	Algebraic expression
Selection	$\sigma(mp, prd) : \{x \mid x \in mp \wedge prd(x)\}$
Map-Window	$\varpi(mp, wn) : \{x \mid central - gravity(x) \in wn \wedge prd(x)\}$
Map-Point	$\pi(mp, pnt) : \{x \mid pnt \in x \wedge prd(x)\}$
Join	$\bowtie (prd, out, mp_1, mp_2) : \{out(x) \mid x_1 \in mp_1 \wedge x_2 \in mp_2 \wedge prd(x_1, x_2)\}$
Union	$mp_1 \cup mp_2 : \{x \mid x \in mp_1 \vee x \in mp_2\}$
Difference	$mp_1 \Delta mp_2 : \{x \mid x \in mp_1 \wedge x \notin mp_2\}$
Intersect	$mp_1 \cap mp_2 : \{x \mid x \in mp_1 \wedge x \in mp_2\}$
<i>Object operators</i>	
Rotate	$y = rotate(x, d)$
Intersect	$y = intersect(x_1, x_2)$
Convert	$convert(x) = \{x\}$
Insert	$insert(mp, obj) = \{x \mid x \in mp \vee x = obj\}$
Other operators	Left to the application to define them
<i>Aggregate functions</i>	
Distance	$distance(x_1, x_2)$
Central-gravity	$central - gravity(x)$
Size	$sizeof(x)$
Other functions	Left to the application to define them

Table 4.1: Spatial operators

- $distance(x_1, x_2)$ : Calculates the distance between the centers of the two input objects. Obviously the result is a primitive scalar type.
- $sizeof(mp)$ : Calculates the number of objects in the given map ( $mp$ ).
- n-dimensional volume, surface, etc.: We may have functions to calculate the volume, surface or a cross section of an object. In general the function can be for a  $n$  dimensional space.

## 4.5 Predicates

As we saw in previous sections some operators may have a predicate in addition to input data. A predicate is a test of a specific relationship between attributes and values. All predicates are logical combination of the following basic predicates. We categorize them in two groups: Basic aspatial predicates and Basic spatial predicates.

**Basic aspatial predicates** These predicates are a test of relationship between two aspatial attributes. Their syntax is presented in Appendix A.

**Basic spatial predicates** They are a test of spatial relationship between two spatial objects. We have tried to introduce a grammar capable of expressing predicates for a n-dimensional space. See appendix A for an example of the traditional direction relationships (left of, right of, etc.) being replaced with a general n-dimensional single word relationship: *direction*. Based on the input parameter, it is interpreted as one of the directional relationships.

Depending on the application space (Topological space, Euclidean space, metric space, network space) one may add other basic spatial relationships to what we introduced here.

**Definition** A basic predicate containing more than one object variable is a multi-object-variable predicate. As we will see in the next chapter this (being a single-object-variable predicate or a multi-object-variable predicate) will be relevant in query optimization procedure.

## 4.6 Examples

A complete spatial algebra must be able to express queries in every kind of spatial application. At the same time it must be simple and robust (having useful theorems and transformation rules for query interpretation and optimization). Here we bring example queries from chapter 1 but this time expressed in our algebra to test its completeness. We will use the examples through the rest of this and next chapter to explain the query processing steps.

**Example 1** In a GIS (Geographical Information System) that contains the map of a country: Find cities with a population bigger than 10000 situated on a 50 km neighborhood of a fault line.

Let us assume cities are indexed in a map named *City* and fault lines are indexed in a map named *Fault-line* the algebraic expression of the query will be:

$$\bowtie ((distance(\lambda_1, \lambda_2) < 50km), x_1, \sigma(City, (City.APT.population > 10000)), \\ Fault - line)$$

First we do a selection on the *City* then the result is joined by *Fault-line* meeting the asked predicate. We may write other expressions for the query with the same result. The query optimizer is responsible for producing and choosing the best one.

**Example2** In a CAD database containing the map of a building: Show rooms with at least one smoke detector and one fire-fighting boxes in its 20m approach. Evaluate the query in the window of interest.

Let us assume rooms are represented by a map named *Rooms*, Fire-fighting boxes by a map named *Fire-Fight* and smoke detectors by a map named *Smoke-Detector*. The

window of interest is specified by a value of type coordination we represent it by "W".  
The algebraic expression of the query will be:

$$\varpi(\bowtie ((distance(\lambda1, \lambda2) < 20m), x_1, (\bowtie ((distance(\lambda1, \lambda2) < 20m), x_1, Rooms, \\ Smoke - Detector)), Fire - Fight), W)$$

This may be an inefficient expression of the query. The query optimizer will be responsible for producing efficient equivalent of the given query expression (see section 5.3.).

**Example3** In a database containing the neural structure of the brain (A potential future scientific application for spatial databases) find neurons connected to less than 10 other neurons in a specified window.

Let us assume that the neurons are represented by a map named *Neurons* and the window of interest is specified by a value of type coordination we represent it by "W".  
Then the algebraic expression of the query will be:

$$\sigma(Neurons, (sizeof(\bowtie (intersects, x_2, (convert(x)), Neurons)) < 10))$$

## 4.7 Query evaluation graph

QEG is a graph equivalent of an algebraic expression. It is produced by replacing operators and functions of the algebraic expression with their equivalent graph elements as we define in this section. The main application of query graph is in query optimization and QEP generation. QEG is helpful for:

- Expressing and applying the query transformation and optimization rules
- Expressing the dependency of the operations (The output of one operation may be the input of the others) in a query and from there finding their order of evaluation.

- Showing the operations that may be executed in parallel

By applying transformations a QEG is transformed into its more efficient equivalent representation from which a QEP is produced. Each QEG can be translated to at least one QEP. A QEP is a set of primitive operations executed in a specified order which finally leads to the evaluation of the query.

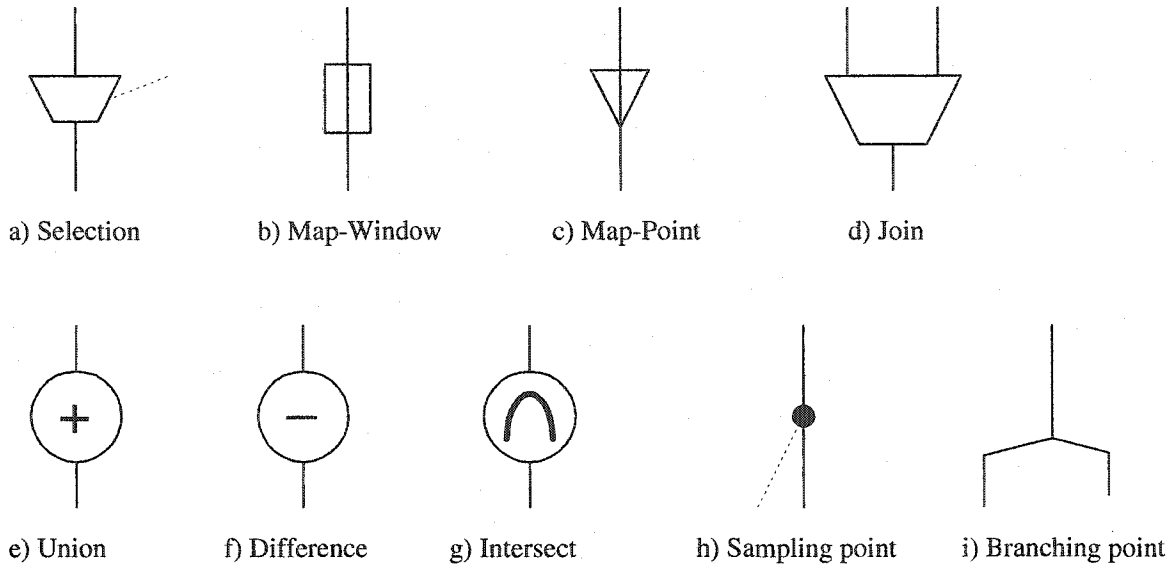


Figure 4.2: Query graph elements

#### 4.7.1 Graph elements

There are nine graph elements as shown in Figure 4.2. Seven of the graph elements represent the seven map operators. A sampling point is shown by a bullet. It represents the nodes that their corresponding object variable is used in the predicate of an operator (Join and Selection). In special case, a Join can be presented by a Selection element and Sampling point as is shown in Figure 4.3. This will help produce more efficient QEPs. As we will see in chapter 5 sampling point has a crucial application in our optimization algorithms. A branching point is used when the output of an operation is the input to more than one operators.

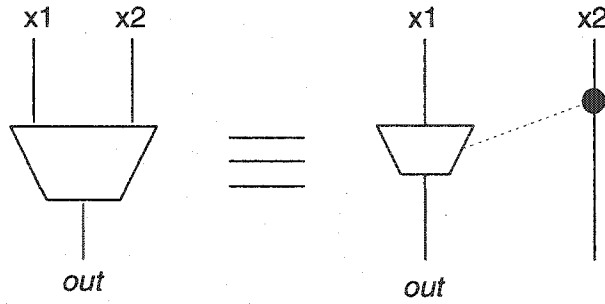


Figure 4.3: An equivalent graph for Join (iff  $out = x_1$ )

#### 4.7.2 Example graphs

Here we bring a graph representation of the examples presented in previous section.

**Example1** The query in Example1 has two operators: one Selection and one Join. This will produce the QEG in Figure 4.4. The graph has two entries and one end ( $Q$ ). The entries represent *City* and *Fault-Line* maps. The ending point ( $Q$ ) represents the query result.

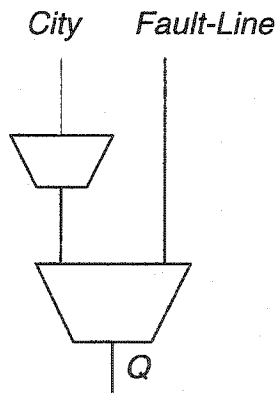


Figure 4.4: QEG for Example 1

**Example2** In this example we have two Join operator and one Window operator (see Figure 4.5) . The inputs to the graph are *Rooms*, *Smoke – Detector* and *Fire – Fight*

maps. First *Rooms* and *Smoke – Detector* do a Join. Then the result does another Join with *Fire – Fight*. Finally the result of this Join passes the Window operator, producing the query result at the end. We were able to represent Join operators with their equivalent (Selection operator and Sampling point). It would lead to a better QEP. In chapter 5 we will see how the query optimizer transforms the QEG in this example to a more efficient equivalent.

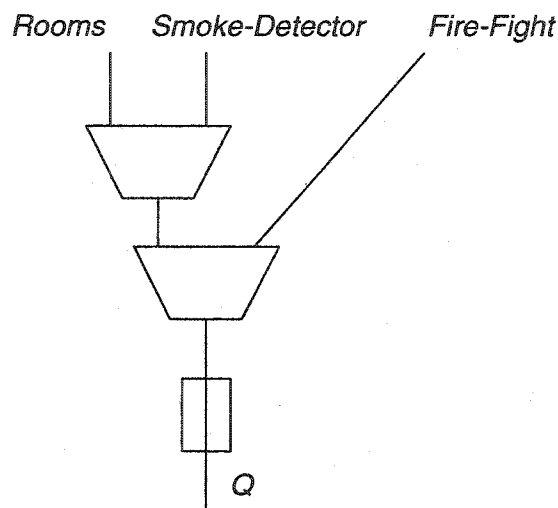


Figure 4.5: QEG for Example 2

**Example3** The graph in this example has two parts(see Figure 4.6). One representing the main query, the other representing the sub-query needed to evaluate the predicate in the main query. The Selection with Sampling on itself represents the Selection operator in main query. The reason for sampling point is that the predicate uses a sub-query that makes a Join with input to the Selection operator (*Neurons*) itself. The graph representing the sub-query used in the predicate, consist of only one Join element. The inputs to sub-graph are *Neurons* map and a map containing only one object (current object that the predicate is evaluated against) of *Neuron*. The result of this sub-graph (sub-query) is changed to a value through the aggregate function *sizeof*. This value will

be used in the main query predicate.

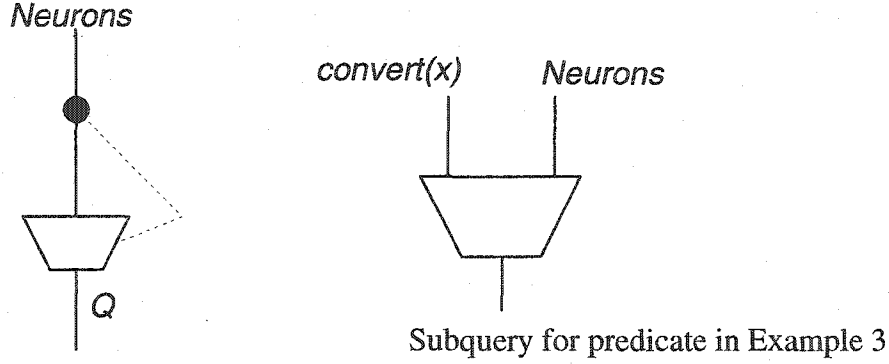


Figure 4.6: QEG for Example 3

## 4.8 Transformation rules

Based on the operators and data types we defined in previous sections we may infer many equations as in the following. Those equations that their right side is less costly than their left side are Transformation Rules (be brought in chapter 5).

The predicate for Selection operator in the following equations is assumed to be single-object-variable predicate:

$$\sigma(\sigma(mp, prd_1), prd_2) \equiv \sigma(\sigma(mp, prd_2), prd_1) \equiv \sigma(mp, prd_1 \wedge prd_2) \quad (4.8.1)$$

$$\varpi(\sigma(mp, prd), W) \equiv \sigma(\varpi(mp, W), prd) \quad (4.8.2)$$

$$\pi(\sigma(mp, prd), pnt) \equiv \sigma(\pi(mp, pnt), prd) \quad (4.8.3)$$

$$\varpi(\varpi(mp, W_1), W_2) \equiv \varpi(\varpi(mp, W_2), W_1) \equiv \varpi(mp, W) \quad (4.8.4)$$

Where  $W$  is the smaller one of  $W_1$  and  $W_2$

$$\pi(\pi(mp, pnt_1), pnt_2) \equiv \pi(\pi(mp, pnt_2), pnt_1) \quad (4.8.5)$$

$$\pi(\varpi(mp, W), pnt) \equiv \varpi(\pi(mp, pnt), W) \quad (4.8.6)$$

$$\pi(\cup(mp_1, mp_2), pnt) \equiv \cup(\pi(mp_1, pnt), \pi(mp_2, pnt)) \quad (4.8.7)$$

$$\pi(\cap(mp_1, mp_2), pnt) \equiv \cap(\pi(mp_1, pnt), \pi(mp_2, pnt)) \quad (4.8.8)$$

$$\pi(\Delta(mp_1, mp_2), pnt) \equiv \Delta(\pi(mp_1, pnt), \pi(mp_2, pnt)) \quad (4.8.9)$$

$$\sigma(\cup(mp_1, mp_2), prd) \equiv \cup(\sigma(mp_1, prd), \sigma(mp_2, prd)) \quad (4.8.10)$$

$$\sigma(\cap(mp_1, mp_2), prd) \equiv \cap(\sigma(mp_1, prd), \sigma(mp_2, prd)) \quad (4.8.11)$$

$$\sigma(\Delta(mp_1, mp_2), prd) \equiv \Delta(\sigma(mp_1, prd), \sigma(mp_2, prd)) \quad (4.8.12)$$

$$\varpi(\cup(mp_1, mp_2), W) \equiv \cup(\varpi(mp_1, W), \varpi(mp_2, W)) \quad (4.8.13)$$

$$\varpi(\cap(mp_1, mp_2), W) \equiv \cap(\varpi(mp_1, W), \varpi(mp_2, W)) \quad (4.8.14)$$

$$\varpi(\Delta(mp_1, mp_2), W) \equiv \Delta(\varpi(mp_1, W), \varpi(mp_2, W)) \quad (4.8.15)$$

$$\cap(\cap(mp_1, mp_2), mp_3) \equiv \cap(\cap(mp_1, mp_3), mp_2) \quad (4.8.16)$$

$$\cup(\cup(mp_1, mp_2), mp_3) \equiv \cup(\cup(mp_1, mp_3), mp_2) \quad (4.8.17)$$

$$\cap(mp_1, mp_2) \equiv \cap(mp_2, mp_1) \quad (4.8.18)$$

$$\cup(mp_1, mp_2) \equiv \cup(mp_2, mp_1) \quad (4.8.19)$$

$$\cap(\cup(mp_1, mp_2), mp_3) \equiv \cup(\cap(mp_1, mp_3), \cap(mp_2, mp_3)) \quad (4.8.20)$$

$$\cup(\cap(mp_1, mp_2), mp_3) \equiv \cap(\cup(mp_1, mp_3), \cup(mp_2, mp_3)) \quad (4.8.21)$$

We prove only one equation (equation 4.8.2). The rest can be proven in a similar way.

**Proof for equation 4.8.2** : We prove every object in the left side will be in the right side and vice versa.

$$\forall \lambda \in \varpi(\sigma(mp, prd), W)$$

Applying definition of  $\varpi$  operator:  $\lambda \in \sigma(mp, prd) \wedge \lambda \in W$

Applying definition of  $\sigma$  operator:  $\lambda \in mp \wedge prd(\lambda) \wedge \lambda \in W$

Then:  $\lambda \in mp \wedge \lambda \in W \wedge prd(\lambda)$

Applying definition of  $\varpi$  operator:  $\lambda \in \varpi(mp, W) \wedge prd(\lambda)$

Applying definition of  $\sigma$  operator:  $\lambda \in \sigma(\varpi(mp, W), prd)$

Now we go from right side of the equation 2 to the left side:

*forall*  $\lambda \in \sigma(\varpi(mp, W), prd)$

Applying definition of *sigma* operator:  $\lambda \in \varpi(mp, W) \wedge prd(\lambda)$

Applying definition of  $\varpi$  operator:  $\lambda \in mp \wedge \lambda \in W \wedge prd(\lambda)$

Then:  $\lambda \in mp \wedge prd(\lambda) \wedge \lambda \in W$

Applying definition of  $\sigma$  operator:  $\lambda \in \sigma(mp, prd) \wedge \lambda \in W$

Applying definition of  $\varpi$  operator:  $\lambda \in \varpi(\sigma(mp, prd), W)$

The proof is complete.

We don't bring transformation rules for Join operator and multi-object-variable Selection operator because their algebraic expression is complex and it is more useful to express them as QEG transformation rules (to be brought in next chapter).

## 4.9 Conclusion

In this chapter we introduced a new algebra (map algebra) for spatial databases. The Algebra and data model is indeed the user view of the database and it is used for the

following main purposes:

1. To organize data in a model that lets efficient query processing
2. To write advanced queries (based on complex predicates) and evaluate them efficiently

It must be at the same time both general (complete) and efficient to be used for a variety of different application in an efficient way.

Some of the characteristics of our model and algebra are:

- It is a very simple model.
- Can express different kinds of queries for different applications.
- It may be combined to a relational, object-relational or object-oriented paradigm to handle non spatial data as well.

Map algebra may be seen as similar to relational algebra or as an spatial extension to relational model but it is a different one. If we compare tuples to objects and relations to maps, the map algebra is different to relational algebra in the following points:

1. The relationship among maps is very simple (they all share the same space) but the relationship among Relations is defined by Primary keys and Foreign keys.
2. The relationship among tuples is very simple (they are in the same relation) but in map algebra each object has a different (spatial) relationship to each of the objects in the same map.
3. A map corresponds to at least one spatial index (R-tree, kd-tree, etc.) and the criteria in the definition of maps is the spatial relationship among objects. In relational model the Relations are defined based on the relationship among attributes.

4. In the map algebra objects are kept intact. While in relational algebra tuples change during algebraic operations (*Division* or *Multiplication*).

Some work is done for the application of a deductive approach towards spatial data. This can be seen as a higher level tool that may be applied for spatial reasoning and query rewriting but once that rewritten query is put to the database the application of our algebra and optimizer starts. There isn't a contradiction or redundancy between our approach (map model) and a deductive approach.

## Chapter 5

# THE SPATIAL QUERY OPTIMIZER

In this section we explain the proposed query optimization and processing steps. By using an example query and passing it through different steps of optimization and processing we see what is the exact procedure of each step. As in the Figure 5.1 our optimizer consist of two main blocks:

1. QEG generation and optimization
2. QEP generation and optimization

In a typical spatial database, after receiving the query from user interface (either in SQL like language or other formats) in the first step (primary interpretation), an algebraic expression of the query is generated. Primary interpretation uses almost the same concepts and methods for spatial and non-spatial systems. Our work doesn't include primary optimization step but we bring here what might have been done to a query before getting to our query optimizer.

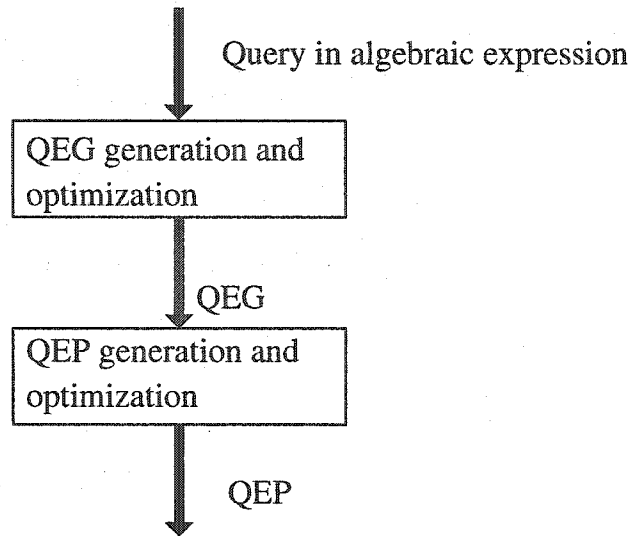


Figure 5.1: Spatial query optimization system

## 5.1 Primary interpretation

The user may put query to a spatial database in SQL like language or through interaction with a high level user interface that is translated to a query in SQL like language. During the primary interpretation of the query one or more of the following tasks are performed:

- **Lexical and syntactical analysis:** In this step the parse tree of the query is constructed, and lexical and syntactical errors are detected according to the grammar of the language.
- **Semantic analysis (to detect impossible or redundant queries):** Meaningless, redundant and impossible queries are detected in this step. Semantic errors are detected by applying semantic rules of the language.
- **Predicate standardization and simplification:** The logical combination of predicates is transformed and rewritten in the simplest form using logical transformation methods. Removing NOT operators or pushing it down as much as possible is done in this step.

- Translation to Algebraic expression

As we see, the primary interpretation for spatial queries uses the same concepts as primary interpretation for conventional queries.

## 5.2 QEG generation

This block receives the algebraic expression of the query and produces one *efficient* QEG. This is done through the following steps:

1. Graph construction: A query graph is constructed directly from the algebraic expression.
2. Graph transformation: In a QEG we can transform the graph by replacing more efficient equivalents of graph elements. The sampling and branching nodes (if there are any) must remain unchanged. This is why we break the graph to sub-graphs on the sampling and branching nodes and transform only the subgraphs.

### 5.2.1 Graph construction

For every query expression there is at least one query equivalent graph with multiple inputs and single output. Inputs (starting nodes) to the graph are map(s) and a single output (ending) node is the query result. The graph is constructed as a parse-tree. Each node of the graph is the result of an operation on one or many of the nodes from the previous level thus the graph can be represented as follows:

$$Q = G_{11} \text{ opr } G_{12}$$

$$G_{11} = G_{21} \text{ opr } G_{22}$$

$$G_{12} = \dots$$

$$\cdot \quad \cdot \quad \cdot$$

$$\cdot \quad \cdot \quad \cdot$$

$$G_{ij} = G_{(i-1)1} \text{ opr } G_{(i-1)2}$$

$$\cdot \quad \cdot \quad \cdot$$

$$\cdot \quad \cdot \quad \cdot$$

where:

$i$  is the distance to the output (query) node

$j$  enumerates the nodes with the same distance

We must also specify if a node is a sampling node. A *Join* or a *Selection* operator that has an object variable in its predicates has a sampling node as its input (the input that corresponds to the object variable).

**Optional optimization-1** To detect and remove redundancy in the graph we can compare the algebraic expression of each node ( $G_{ij}$ ). If there are nodes with the same algebraic expression we can replace them with only one node. This node will be a branching node.

We add an attribute to each node to show its type:

1. Normal node
2. Sampling node
3. Branching node

Also when generating QEP we need other attributes to represent the state of evaluation of a node. During the evaluation of a query, the nodes of its QEG will be in one of the following three states:

1. not evaluated
2. approximately evaluated
3. exactly evaluated

This attribute is used in QEP generation algorithm (to be brought in next section).

### 5.2.2 Graph transformation

The QEG transformation is done by applying transformation rules to the QEG components. Transformation rules are equations having their right side being *always* less costly than their left side. There are some uncertain transformations that do not necessarily lead to a better QEP. We don't study them here. In a more complete optimizer they may be used to produce many QEP and select the best one based on cost evaluation. Here we introduce the transformation rules:

1.  $\sigma(\sigma(mp, prd_1), prd_2) \equiv \sigma(mp, prd_1 \wedge prd_2)$
2.  $\varpi(\sigma(mp, prd), W) \equiv \sigma(\varpi(mp, W), prd)$
3.  $\pi(\sigma(mp, prd), pnt) \equiv \sigma(\pi(mp, pnt), prd)$
4.  $\varpi(\varpi(mp, W_1), W_2) \equiv \varpi(\varpi(mp, W_2), W_1) \equiv \varpi(mp, W)$

Where  $W$  is the smaller one of  $W_1$  and  $W_2$

5.  $\pi(\varpi(mp, W), pnt) \equiv \varpi(\pi(mp, pnt), W)$

6.  $\cup(\cap(mp_1, mp_3), \cap(mp_2, mp_3)) \equiv \cap(\cup(mp_1, mp_2), mp_3)$
7.  $\cap(\cup(mp_1, mp_3), \cup(mp_2, mp_3)) \equiv \cup(\cap(mp_1, mp_2), mp_3)$
8. A *Join* operation in special cases is equivalent to a *Selection* operation (with a sampling node on its second object variable).

In all of the above transformations, evaluating the right side is *always* less expensive than the left side (the cost of an operation depends on the number of objects in the domain and its implementing primitive operations) so the transformed QEG will be *always* less expensive to evaluate.

When we apply transformation rules to a QEG components we must keep the sampling nodes and branching nodes unchanged because they are used by multiple graph components.

**The QEG transformation algorithm** Regarding the above facts and concepts here is the algorithm that transforms a QEG to a more efficient (less expensive) QEG:

```

Start from Q node (Query node) of the QEG.
for level = 0 to last level of the QEG
{
    for each node in current level
    {
        if input(s) to this node are Normal node(s)
        {
            check if a transformation rule matches
            {
                apply the corresponding transformation
                update the graph
            }
        }
    }
}

```

**Discussion1** We may apply the QEG transformation algorithm to the QEG more than once. As each QEG is more efficient than the previous one, we may continue until the

QEG is saturated and the algorithm can not change the QEG. However, theoretically it may happen after infinite times (depending on the transformation rules).

**Discussion2** Because each QEG transformation produces a more efficient QEG, multiple application of the QEG transformation algorithm won't reproduce a QEG from previous steps.

**Definition** The maximum number of application of QEG transformation algorithm, is a parameter for the optimizer and is shown as  $n_{opt}$ .

**Discussion3** Although in the QEG transformation algorithm we always start sweeping from Q node it is possible to start from any other node. Different patterns of sweeping the QEG may produce different results in theory. This question remains open: What is the optimum pattern of sweeping the QEG (if there is any)? A more complex optimizer may try different sweeping patterns and produce many QEG to be passed to the next step of optimization.

**Discussion4** The QEG transformation algorithm will always produce a finite QEG from a finite original QEG. The reason is that in every step (application of one of the transformation rules) the number of levels either remain the same or is reduced. Also the number of nodes of the current level may be increased only to the maximum of number of nodes in the upper level.

### 5.3 QEP generation

A QEP is a set of primitive operations that run in a specified order. The QEP is constructed based on the QEG as a two dimensional array of operations. The first

dimension must run sequentially while the operations in the second dimension may run in parallel. This will be useful in the case of distributed processing. The primitive operations are the possible combination of the following three modes and the algebraic operations. The three operation modes are:

1. Filter
2. Refine
3. Feedback

In Filter mode an operation only runs on an approximate (MBR) of the objects or Basic Predicates are tested approximately. Whatever the approximation is, it must be inclusive. In other words the result of operation in Filter mode must include all objects in the final result. In Refine mode of an operation we work with the exact object and exact evaluation of Basic Predicate rather than an approximation, to refine the filtered result. The approximate evaluation of an operation is much cheaper than the exact evaluation and it reduces the domain for the next phase (exact evaluation), leading to much cost effective evaluation of an operation. The sampling nodes must always be evaluated exactly and we can not use the approximate evaluation of a sampling node for the approximate evaluation of other nodes in the graph, so use Filter and Refine mode sequentially for the evaluation of sampling nodes of the graph. Feedback is used for multilevel filtering/refinement approach. In this mode the approximate results of one level are sent back many levels to prune (filter) the domain (inputs) of the operation even more. We show how these work by an example later in this chapter. The algorithmic representation of these primitive operations are as follows:

```
Filter( node1 )  
{  
    Evaluate the graph element corresponding to node1. Use only  
    the inclusive approximate evaluation (e.g. MBR).
```

```

    Set this node status to "Approximately Evaluated"
}

Refine( node1 )
{
    Evaluate the graph element corresponding to node1. Before
    performing the exact evaluation check if the object belongs to
    the approximate result attached to node1. Otherwise don't
    perform the costly exact evaluation on this object.

    Set the status of this node to "exactly evaluated".
}

Feedback( node1 , node2 )
{
    Intersect the map attached to node1 with the map attached to node2.
    Write the result to map attached to node1. The map attached to
    node2 won't be changed.
}

```

**The QEP generation algorithm** Here we bring the algorithm that constructs an efficient QEP from a QEG. The algorithm is based on using multilevel filter and refinement approach. In this approach we prune (filter) the domain (input maps) of an operation by using the results of approximate evaluation of next levels of the QEG. The algorithm also uses QEG to determine which operations can be executed in parallel.

```

For i = farthest level of the graph to i = 0
{
    In parallel for all j
    {
        If the node is normal
        {
            Filter (approximate evaluation)  $G_{ij}$  and set the
             $G_{ij}$  node to "approximately evaluated".
        }
        Else (the node is a sampling node, branching node or Q node)
        {
            If the input(s) to this node are already "exactly evaluated"
            Filter and Refine this node and set it to "exactly
            evaluated".
            Else
            {
                Filter this node.

                Recursively go back until the exactly evaluated nodes and:
                Feedback this node to the upper level nodes.
                Filter/Refine the nodes forward down to this node.
            }
        }
    }
}

```

Filter/Refine this node and set it to "exactly evaluated".

**Example2** Here we show how the above algorithms are implemented using Example2 of Chapter 4. After applying the graph transformation rules to the the QEG for of the efficient QEG for this example will be as following:

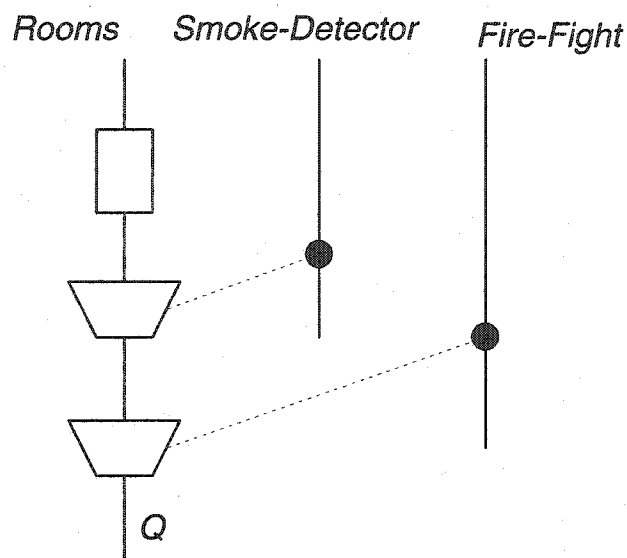


Figure 5.2: QEG for Example 2 after transformation

The QEG before transformation :

$$Q = G_{11} \text{ Map-Window}$$

$$G_{11} = G_{21} \text{ Join } G_{33}$$

$$G_{21} = G_{31} \text{ Join } G_{32}$$

$$G_{31} = Rooms$$

$$G_{32} = \text{Smoke} - \text{Detector}$$

$$G_{33} = \textit{Fire} - \textit{Fight}$$

The QEG after transformation:

$Q = G_{11} \text{ Selection } G_{32}$   
 $G_{11} = G_{21} \text{ Selection } G_{33}$   
 $G_{21} = G_{31} \text{ Map} - \text{Window}$   
 $G_{31} = \text{Rooms}$   
 $G_{32} = \text{Smoke} - \text{Detector}$   
 $G_{33} = \text{Fire} - \text{Fight}$

The QEP constructed from the transformed QEG will be as following:

*Phase1 :*

*Filter(G21)*

*Phase2 :*

*Filter(G11)*

*Phase3 :*

*Filter(G00)*

*Feedback(G21, G00)*

*Refine(G21)*

*Filter(G11)*

*Refine(G11)*

*Filter(G00)*

*Refine(G00)*

### 5.3.1 Spatial predicate evaluation

There are conventional ways to simplify the combinational predicates by using logic algebra. We don't expand our work to this area. We assume that the predicate is already simplified by using conventional methods.

Regarding the following facts about spatial databases the order of evaluation of the

terms of a combinational predicate will be determining in the cost of evaluation of the predicate.

1. As soon as we find a predicate in sum of predicates is *True* we don't need to evaluate the other terms; The result will be *True*.
2. If a predicate in product of predicates is *False* we don't need to evaluate the other terms; The result will be *False*.
3. In evaluation of product of predicates the domain of each predicate (either combinational predicate or basic predicate) is the result of previous predicate and because the cost of evaluation is proportional to the domain, the later we evaluate a basic predicate the less expensive it will be.

To find the most efficient equivalent of a predicate we write all possible permutation of its terms. Then we estimate the cost for each equivalent to choose the least expensive (the most efficient).

**Cost estimation** Assign a cost ( $c_i$ ) and restriction ( $r_i$ ) to each basic predicate in the combinational predicate with the following definition:

$c_i$ : The cost of evaluating the  $i^{th}$  predicate on a single object.

$r_i$ : The possibility that  $i^{th}$  predicate is *True* for an object.

$d_i$ : The domain of  $i^{th}$  predicate. In other words the size of set of objects that this predicate must be evaluated against.

$C_{total}$ : The cost of evaluation of the combinational predicate. By definition it is the sum of the costs of all basic predicates on their respective domains:

$$C_{total} = \sum_{i=1}^n c_i * d_i \quad (5.3.1)$$

Where  $n$  is the number of basic predicates.

Based on the above definitions and by using probability theory we can write the following equations.

For logical product of  $n$  predicates we use the following equation:

$$d_i : 1, r_1, r_1 r_2, \dots, d_{i+1} = d_i r_i \dots \quad (5.3.2)$$

and the total restriction will be:

$$r_{total} = \prod_{i=1}^{n-1} r_i \quad (5.3.3)$$

For logical sum of  $n$  predicates we use the following equation:

$$d_i : 1, 1 - r_1, (1 - r_1)(1 - r_2), \dots, d_{i+1} = d_i(1 - r_i), \dots \quad (5.3.4)$$

and the total restriction will be:

$$r_{total} = 1 - d_{n+1}/d_1 \quad (5.3.5)$$

In the above equations we assume the basic predicates are independent (zero correlation).

Please note that the accuracy of the cost estimation depends on the accuracy of  $r_i$  and  $c_i$  estimation. For example  $c_i$  for **MEET** predicate is much more than  $c_i$  for **direction** predicate because evaluating **MEET** predicate takes much more time than evaluating **direction** predicate.

**Example** Assume that we have the following predicate to be evaluated for a *Selection* or *Join* operation:

$$s_1 s_2 (s_3 + s_4 s_5)$$

Also assume that the order of evaluation of the predicates is from left to right for each written mutation. We may have the following equivalents of the predicate:

$$\text{Equivalent}_1 : s_1 s_2 (s_3 + s_4 s_5)$$

$$\text{Equivalent}_2 : s_1 s_2 (s_4 s_5 + s_3)$$

$$\text{Equivalent}_3 : s_1 s_2 (s_3 + s_5 s_4)$$

$$\text{Equivalent}_4 : s_1 s_2 (s_5 s_4 + s_3)$$

$$\text{Equivalent}_5 : s_2 s_1 (s_3 + s_4 s_5)$$

$$\text{Equivalent}_6 : s_2 s_1 (s_4 s_5 + s_3)$$

$$\text{Equivalent}_7 : s_2 s_1 (s_3 + s_5 s_4)$$

$$\text{Equivalent}_8 : s_2 s_1 (s_5 s_4 + s_3)$$

$$\text{Equivalent}_9 : s_1 (s_3 + s_4 s_5) s_2$$

$$\text{Equivalent}_{10} : s_1 (s_4 s_5 + s_3) s_2$$

$$\text{Equivalent}_{11} : s_1 (s_3 + s_5 s_4) s_2$$

$$\text{Equivalent}_{12} : s_1 (s_5 s_4 + s_3) s_2$$

$$\text{Equivalent}_{13} : s_2 (s_3 + s_4 s_5) s_1$$

$$\text{Equivalent}_{14} : s_2 (s_4 s_5 + s_3) s_1$$

$$\text{Equivalent}_{15} : s_2 (s_3 + s_5 s_4) s_1$$

$$\text{Equivalent}_{16} : s_2 (s_5 s_4 + s_3) s_1$$

$$Equivalent_{17} : (s_3 + s_4 s_5) s_1 s_2$$

$$Equivalent_{18} : (s_4 s_5 + s_3) s_1 s_2$$

$$Equivalent_{19} : (s_3 + s_4 s_5) s_2 s_1$$

$$Equivalent_{20} : (s_4 s_5 + s_3) s_2 s_1$$

$$Equivalent_{21} : (s_3 + s_5 s_4) s_1 s_2$$

$$Equivalent_{22} : (s_5 s_4 + s_3) s_1 s_2$$

$$Equivalent_{23} : (s_3 + s_5 s_4) s_2 s_1$$

$$Equivalent_{24} : (s_5 s_4 + s_3) s_2 s_1$$

and let:

$$C_1 = 50, C_2 = 75, C_3 = 40, C_4 = 20, C_5 = 60$$

$$r_1 = 0.25, r_2 = 0.15, r_3 = 0.35, r_4 = 0.3, r_5 = 0.25$$

and using equations 5.3.1 to 5.3.5 we can calculate the cost estimation for all equivalents as following:

$$Equivalent_1 : C_{total} = 71.177$$

$$Equivalent_2 : C_{total} = 71.562$$

$$Equivalent_3 : C_{total} = 71.834$$

$$Equivalent_4 : C_{total} = 72.575$$

$$Equivalent_5 : C_{total} = 84.926$$

$$Equivalent_6 : C_{total} = 85.312$$

$$Equivalent_7 : C_{total} = 85.584$$

$$Equivalent_8 : C_{total} = 86.325$$

*Equivalent*<sub>9</sub> :  $C_{total} = 71.917$

*Equivalent*<sub>10</sub> :  $C_{total} = 74.492$

*Equivalent*<sub>11</sub> :  $C_{total} = 76.305$

*Equivalent*<sub>12</sub> :  $C_{total} = 81.242$

*Equivalent*<sub>13</sub> :  $C_{total} = 93.472$

*Equivalent*<sub>14</sub> :  $C_{total} = 96.047$

*Equivalent*<sub>15</sub> :  $C_{total} = 97.859$

*Equivalent*<sub>16</sub> :  $C_{total} = 102.80$

*Equivalent*<sub>17</sub> :  $C_{total} = 85.755$

*Equivalent*<sub>18</sub> :  $C_{total} = 96.055$

*Equivalent*<sub>19</sub> :  $C_{total} = 91.497$

*Equivalent*<sub>20</sub> :  $C_{total} = 101.80$

*Equivalent*<sub>21</sub> :  $C_{total} = 103.30$

*Equivalent*<sub>22</sub> :  $C_{total} = 123.05$

*Equivalent*<sub>23</sub> :  $C_{total} = 109.05$

*Equivalent*<sub>24</sub> :  $C_{total} = 128.80$

Comparing the cost estimation we find the *Equivalent*<sub>1</sub> is the most efficient.

**Approximate evaluation** For each spatial predicate we can prune the search space by evaluating corresponding predicates on the spatial approximation of the objects (e.g. MBR). In table 5.1 we present a set of predicates and their corresponding approximation. There are two types of approximation for each predicate: rejecting and approving. The

Spatial predicate	Approving approximation	Rejecting approximation
$A \text{ DISJOINT } B$	$\Box A^1 \text{ DISJOINT } \Box B^1$	N/A
$A \text{ CONTAINS } B$	$A \text{ CONTAINS } \Box B$	$\Box A \text{ DISJOINT } \Box B$ $\Box A \text{ OVERLAP } \Box B$
$A \text{ INSIDE } B$	$\Box A \text{ INSIDE } B$	$\Box A \text{ DISJOINT } \Box B$ $\Box A \text{ OVERLAP } \Box B$
$A \text{ EQUAL } B$	N/A	$\Box A \text{ all other relationships } \Box B$
$A \text{ MEET } B$	$\Box A \text{ MEET } \Box B$	$\Box A \text{ DISJOINT } \Box B$
$A \text{ COVERS } B$	N/A	$\Box A \text{ DISJOINT } \Box B$ $\Box A \text{ OVERLAP } \Box B$
$A \text{ COVERED BY } B$	N/A	$\Box A \text{ DISJOINT } \Box B$ $\Box A \text{ OVERLAP } \Box B$
$A \text{ OVERLAP } B$	N/A	$\Box A \text{ DISJOINT } \Box B$

Table 5.1: Spatial predicate approximation

approximate predicates (both rejecting and approving) take much less time and cost to be evaluated than the main predicate itself. The rejecting predicate is used in *Filter* operations of the QEP. The approving predicate is used in *Refine* operations of the QEP. If the rejecting predicate is *true* for an objects then the main predicate is *false* for that object and the object doesn't belong to the result so it can be filtered out. If the approving predicate is *true* for an object then the main predicate is *true* for that object and the object belongs to the result. Table 5.1 represents only topological relationships but it can be expanded to cover other spatial relationships (e.g. neighborhood and order relationships).

## 5.4 Conclusion

We introduced a query optimization and query processing system using our algebra (the map algebra we introduced in chapter 4). The new concepts and techniques in spatial query processing that we invented and introduced here are as follows:

- Multi-level filter and refinement: Although the Filtering and refinement in spatial query processing is a known technique [HWANG, 1994] but here we apply it in a more efficient way (multi-level) and then feedback the result for refinement.
- A QEG technique: It is used as a tool for the following purposes at the same time:
  1. Recognize and eliminate redundancy in the query expression
  2. Multi-level filtering and refinement
  3. Application of query transformation rules
  4. Generating efficient QEP while having the possibility of distributed and parallel processing of the query (all nodes with the same distance to the source node can be processed in parallel).
- Predicate ordering: In a combinational predicate the order of evaluating sub predicates is a determinant factor in the cost of evaluation. We introduced a cost estimation approach to estimate the cost of equivalent expressions of a combinational predicate and choose the most efficient one.

---

<sup>1</sup>A bounding polygon of the object (an approximation of the spatial object). Most of the spatial indexing techniques use Minimum Bounding Rectangle as an approximation of the spatial object.

## Chapter 6

# EXPERIMENTAL RESULTS

To evaluate and test the introduced query processing and optimization algorithms, a query processing system was implemented in a simulated environment. In this chapter different modules of the implemented system are explained and the experimental results are discussed.

### 6.1 The implemented query optimizer

The query processing system is implemented in Java and consist of many classes to provide the following:

- Random query generator
- QEG generator (with and without the optimization)
- QEP generator (with and without the optimization)
- Spatial database simulator to estimate the processing cost of QEP

### 6.1.1 Random query generator

The random query generator generates queries based on the following input parameters:

1. Maximum number of nesting levels
2. Maximum number of maps in the spatial database

The Java Random class is used to generate random queries. Those methods of random generator that produce a uniformly distributed result are used. The query generator starts from the root of the query tree and builds it level by level. In each level it calls the random generator to produce one operator out of seven possible operators. It continues to build all the needed operators for that level, then goes to the next level until reaches the given maximum number of levels. A Null-Operator is defined to simulate a jump to upper level so even the number of nesting levels becomes a stochastic variable. The input to random query generator is the maximum number of levels not the number of levels itself. The generated query tree is passed to the QEG generator and optimizer.

### 6.1.2 QEG generator and optimizer

The QEG handling class contains the attributes and methods to do the following tasks:

1. Construct QEG from a query tree
2. Transform QEG using the algorithm introduced in chapter 5

QEG constructor receives a query tree in input and builds the QEG data structures. The main data structure of this class is a two dimensional structure (Vector of arrays). The first dimension represents the levels of the QEG and second dimension enumerates

the operators on the same level of the graph. The level of an operator is its longest path to the query node on the QEG. This data structure is used later by the QEP that has a three dimensional data structure. The class also contains many methods that do the following tasks as explained in chapter 5:

1. Remove redundancy in QEG
2. Predicate ordering (for SELECTION and JOIN operators)
3. QEG transformation

The generated (and transformed) QEG is passed to QEP generator and optimizer.

### 6.1.3 QEP generator and optimizer

The QEP handling class contains methods and attributes to construct a QEP from a QEG and to optimize the generated QEP using the multi-level filter/refinement technique introduced in chapter 5. The main data structure of this class is a three dimensional data structure (Vector of Vector of Vectors). Two dimensions of this structure are in parallel to the QEG data structure. The third dimension enumerates the set of primitive operations that implement the corresponding QEG element. The primitive operations are of the following types:

1. Filter
2. Refine
3. Feedback

These primitive operation types are explained in the next subsection.

All vectors on second dimension can be evaluated simultaneously. This will be helpful when parallel query processors are available. Our experiments are done on a single processor system.

The QEP is passed to QEP processor simulator.

#### 6.1.4 Simulator of spatial primitive operation processor

It executes the primitive operations on the simulated spatial database and gives back the estimated cost of QEP evaluation. The maps in the simulated database have only one attribute: number of spatial objects in the map that we refer to it as map size. The cost of each primitive operation is estimated based on the following parameters:

1. The type of the operation itself (Filter, Refine, Feedback)
2. The corresponding QEG graph element
3. The size of the map that the operation is performed on

The Filter, Refine and Feedback operations are implemented according to their definitions in chapter 5. They update the size of the map attached to the corresponding QEG element and return the estimated cost based on the size of the input maps to that QEG element. To estimate the restriction associated to predicates and operators, a random generator is used that generates uniformly distributed numbers in the range of 0,1.0. Also the following items are assumed:

1. The cost of operations is proportional to the size of the input maps. The reason is that in every operation a test on all individual spatial objects is involved.
2. The cost of Filter operation is much smaller than the cost of Refine operation. The reason is that in Filter operation MBR of the objects are used rather than the

objects themselves so the tests are very simple. In Refine operations the complete spatial objects are involved so the tests can be complex.

## 6.2 Experiments

We put randomly generated queries to the optimizer to generate QEP with and without optimization. Then we put the generated QEP to the simulated spatial database and get back the estimated processing costs. To compare the effect of each optimization technique on the processing cost, the following experiments were carried out using 200000 randomly generated queries. All experiments were done on a vast range of input parameters but here we show the results on a selected range, enough to discuss the characteristic of the system. The input parameter values and ranges are as following:

- Maximum number of nesting levels of the randomly generated queries:  $n_{nest} = 2, 4$
- Maximum number of database maps involved in the queries:  $n_{mp} = 5, 10, 15, 20, 25$
- Average number of spatial objects in the maps:  $n_{obj} = 1000$

For example randomly generated queries and their QEP before and after applying the optimization techniques refer to Appendix B.

**Experiment1: Processing queries without optimization** In this experiment the queries are processed without applying any optimization technique. The processing cost of the generated QEPs were estimated by the QEP processor simulator. The results of this experiment are used as a reference for the next experiments to calculate the cost improvement factor (the ratio of the saved cost to the total cost).

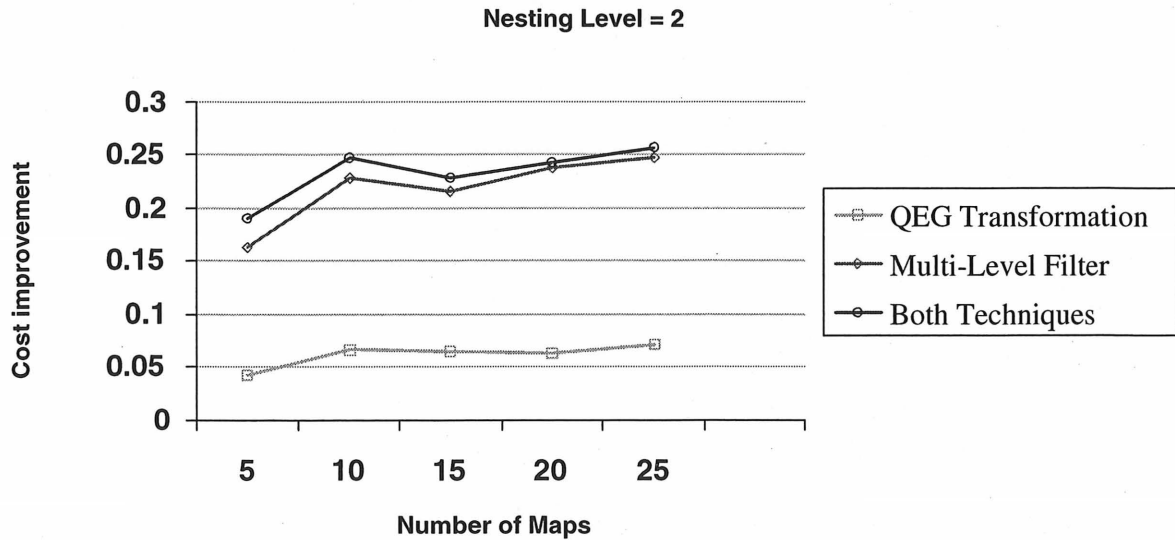


Figure 6.1: Cost improvement for queries with nesting level = 2

**Experiment2: Processing queries applying QEG transformation** In this experiment the transformation rules of chapter 5 are applied to the QEG. As we see this will save 5 to 10 percent of the processing cost depending on the number of maps and query nesting level. The set of transformation rules applied in this experiment did not include all possible useful transformation rules. One may try with a more complete set of transformation rules and achieve a better result.

Figure 6.1 and Figure 6.2 show the result of this experiment in green.

### Experiment3: Processing queries applying multi level Filter and Refinement

In this experiment the multi-level filter/refinement technique was applied to the QEPs. As we see the cost improvement from this technique is in the range of 15 to 30 percent depending on the number of input maps and nesting level. The cost improvement seems to have a saturation value of around 30 percent as the experiment results show, even for a wider range of input maps and nesting levels.

Figure 6.1 and Figure 6.2 show the result of this experiment in red.

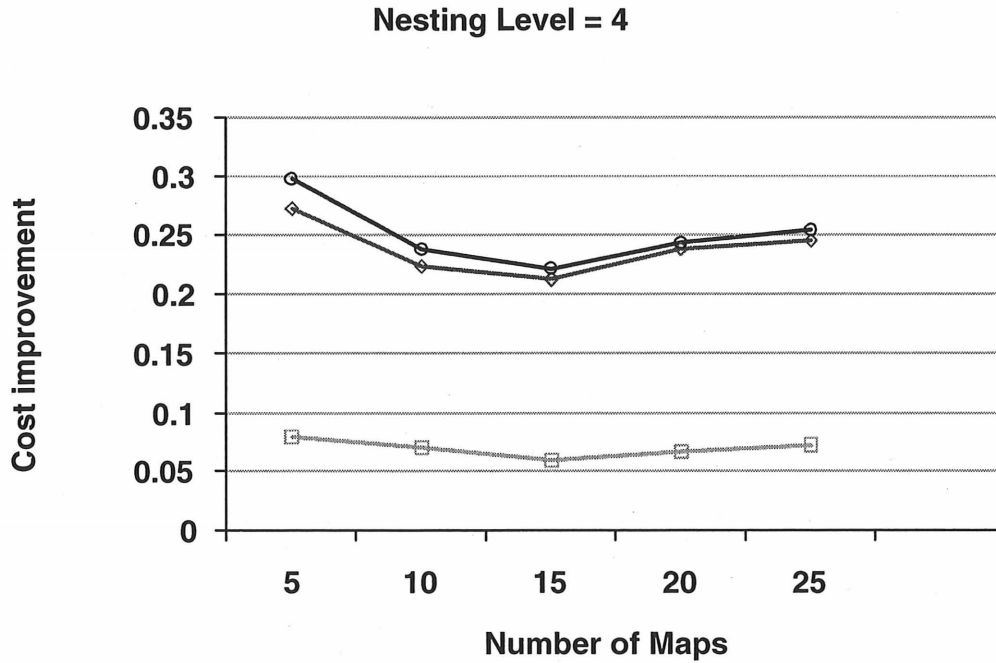


Figure 6.2: Cost improvement for queries with nesting level = 4

**Experiment4: Processing queries applying both techniques** One may expect that applying both optimization techniques (QEG transformation and multi-level filter/refinement) will compound the effect of cost improvement. As the Figure 6.1 and Figure 6.2 show (the result of this experiment is in blue) for low number of input maps this is true but for higher number of input maps the compounding is not very perfect. It seems the two techniques are not completely independent. For more detailed experiment results refer to Appendix B.

### 6.3 conclusion

A complete query processing and optimizer was implemented. To test its efficiency and extract some of its characteristic the generated QEPs were put to a spatial database

simulator. The estimated costs shows that the optimizer in full optimization can improve query processing cost by more than 30 percent. Multi-level filter/refinement seems to be more effective than the QEG transformation. Depending on the nesting level, and number of input maps the effect of two techniques may sum up.

# CONCLUSION

After an investigation of the current approaches for query representation and processing in spatial databases it was concluded that a new model and algebra is needed to express queries and optimization procedures in a more complete and efficient way. To that end the following items were accomplished during this work:

1. A new map algebra was introduced. To show the completeness of the algebra, different example queries from different applications were expressed in the new model. The set of seven operators and the set of spatial relationships (both topological and order relationships) have given the algebra the ability to express queries for a wide range of applications. Also many useful equations were presented that were used in the query transformation. Depending on the application one may add more transformation rules. The experiment was carried out only with a limited set of rules to show how the system works and behaves. (chapter 4).
2. A query processing procedure was introduced. The innovative methods of transforming the query and optimizing it through several steps (algebraic expression, QEG, QEP) until obtaining the optimized QEP was based mainly on the introduced map model. A QEG approach provided a platform for implementation of new techniques such as multi-level filter/refinement. Also in future works it can be used for distributed query processing as it shows the operations that can be executed in parallel. One other application of the QEG is that it is used to write down

the algebraic expression of each node of the QEG to find and remove redundancy in the query. (chapter 5).

3. The filter/refinement procedure that is the core of every spatial query processing system was improved to span over many levels in the case of nested queries. The new technique named multi-level filter/refinement showed a surprising performance in the experimental results (chapter 6 and section 5.3).
4. Regarding the nature of spatial data and the fact that the order of evaluation of sub-predicates is relevant in the overall cost, we introduced a predicate ordering method. The method is based on the cost estimation for all possible equivalent orders of basic predicates in a combinational predicate and then choosing the most efficient one (subsection 5.3.1).
5. A spatial predicate approximation table was introduced (Table 5.1). It can be used in executing a QEP. The approximation can filter the domain of an operation with a very small cost compared to the cost of exact evaluation.
6. A query processing system was designed and implemented based on the above concepts and approaches. Randomly generated queries were put to the system and the results were analyzed for different combination of input parameters. The charts in chapter 6 show the result of the experiments. The system in the simulated environment can reduce the cost to 0.7 of its original cost by applying the introduced optimization techniques. Also we can see that for nesting level = 2 the optimizer becomes more efficient with the increase in the number of maps in the database. In general we can say this query optimizer will be more efficient with bigger databases.

There are other approaches towards spatial databases that can be used as complementary part to our system. For example spatial reasoning and deductive approach may

transform the query before reaching our optimizer. There have been some efforts to use fuzzy logic in spatial databases. Such a system will need fuzzy indexing techniques and no fuzzy indexing techniques so far is available or even definable. If a fuzzy system is to be implemented it can use some of the concepts developed here such as multi-level filter/refinement and spatial predicate approximation (Table 5.1).

We investigated and elaborated only on spatial part of data. Lots of works have already been done on aspatial issues. If a real world application is going to use our system it can handle aspatial attributes as well but the organization of the database and the design of maps will be based on the spatial attribute of the objects. The user can write queries based on the aspatial relationship of attributes but there is no operator equivalent to Production of two Relations like the one in Relational algebra.

The work can continue to elaborate the discussion<sup>1</sup> to discussion<sup>4</sup> in chapter 5. Also for multi-processor systems or distributed databases we may continue to develop algorithms to schedule the operations in QEP. As mentioned in chapter 6 the QEP structure is three dimensional and has the potential to implement scheduling algorithms for more complex systems.

# Appendix A

## A BNF Grammar For The Spatial Algebra

In this section we present a BNF grammar for the spatial algebra. The non-terminal expressions are written in *slanted* font. Terminal expressions are written as following:

- **Keywords** They represent operators or other signs that are not translated. They are written in **this** font.
- **mapidentifier** It represents the name of a map.
- **functionidentifier** It represents the name of a function.
- **aggregate-function** It represents an aggregate function.
- **CONSTANT** It represents an alphanumeric or a spatial constant.

**How to read the grammar** Read the grammar as following:

→ Reads like the left part is defined by (could be written as) the right part.

ε Means empty.

| Means or.

For example the following expression:

*predicate* → ε  
                  | *aspatial-predicate*  
                  | *spatial-predicate*

is read as:

*predicate* is empty or it should be a *aspatial-predicate* or a *spatial-predicate*.

### The grammar

$query \rightarrow selection$

|  $join$

|  $set-operation$

|  $map-window$

|  $map-point$

|  $mapidentifier$

$selection \rightarrow \sigma(map, predicate)$

$join \rightarrow \bowtie(predicate, output, map, map)$

$set-operation \rightarrow set-operator(map, map)$

$set-operator \rightarrow \cup$

|  $\Delta$

|  $\cap$

$map-window \rightarrow \varpi(map, window)$

$map-point \rightarrow \pi(map, point)$

$map \rightarrow query$

| *mapidentifier*

*window*  $\rightarrow$  *value*

*point*  $\rightarrow$  *value*

*output*  $\rightarrow \epsilon$

|  $x_1$

|  $x_2$

|  $x_1 + x_2$

*value*  $\rightarrow$  **CONSTANT**

| *aggregate-function*

*predicate*  $\rightarrow \epsilon$

| *aspatial-predicate*

| *spatial-predicate*

| *NOT predicate*

| *predicate AND predicate*

| *predicate OR predicate*

*aspatial-predicate*  $\rightarrow$  *value aspatial-relationship value*

*aspatial-relationship*  $\rightarrow <$

|  $>$

|  $==$

| <=

| >=

*spatial-predicate* → value *spatial-relationship* value

*spatial-relationship* → *topological-relationship*

| *directional-relationship*

*topological-relationship* → **INSIDE**

| **CONTAINS**

| **EQUAL**

| **DISJOINT**

| **MEET**

| **COVERS**

| **COVERED BY**

| **OVERLAP**

*directional-relationship* → **direction**(value )

### Semantic rules

- Predicates of Join operator must have two variables.
- Predicates of Selection operator have one or two variables.

# Appendix B

## Experimental Results

### B.1 Average estimated costs and improvements

Here we bring the results of the experiments on 200000 randomly generated queries. **cost1** represents the estimated average processing cost of a query when no optimization technique is used, **cost2**: when we have only QEG optimization, **cost3**: when we have only multi-level filter/refine optimization and **cost4**: when we have both techniques (QEG transformation and multi-level filter/refine). The improvement of each optimization technique was calculated as follows:

improvement by QEG transformation =  $(\text{cost1} - \text{cost2}) / \text{cost1} = 1 - \text{cost2} / \text{cost1}$

improvement by Multi-Level Filter/Refine =  $(\text{cost1} - \text{cost3}) / \text{cost1} = 1 - \text{cost3} / \text{cost1}$

improvement by both =  $(\text{cost1} - \text{cost4}) / \text{cost1} = 1 - \text{cost4} / \text{cost1}$

- Queries with: nesting level = 2, number of maps = 5  
cost1: no optimization = 4.54E8  
cost2: only QEG transformation = 4.37E8  
cost3: only Multi-Level Filter/Refinement = 3.94E8  
cost4: full optimization = 3.93E8  
improvement by QEG transformation = 1.0 - 0.963  
improvement by Multi-Level Filter/Refine = 1.0 - 0.868  
improvement by both = 1.0 - 0.867
- Queries with: nesting level = 4, number of maps = 5  
cost1: no optimization = 4.62E8

- cost2: only QEG transformation = 4.45E8
- cost3: only Multi-Level Filter/Refinement = 4.00E8
- cost4: full optimization = 4.0E8
- improvement by QEG transformation = 1.0 - 0.962
- improvement by Multi-Level Filter/Refine = 1.0 - 0.865
- improvement by both = 1.0 - 0.865
- Queries with: nesting level = 2, number of maps = 10
  - cost1: no optimization = 1.33E9
  - cost2: only QEG transformation = 1.25E9
  - cost3: only Multi-Level Filter/Refinement = 1.07E9
  - cost4: full optimization = 1.06E9
  - improvement by QEG transformation = 1.0 - 0.940
  - improvement by Multi-Level Filter/Refine = 1.0 - 0.808
  - improvement by both = 1.0 - 0.800
- Queries with: nesting level = 4, number of maps = 10
  - cost1: no optimization =1.34E9
  - cost2: only QEG transformation=1.26E9
  - cost3: only Multi-Level Filter/Refinement =1.08E9
  - cost4: full optimization =1.07E9
  - improvement by QEG transformation = 1.0 - 0.940
  - improvement by Multi-Level Filter/Refine = 1.0 - 0.807
  - improvement by both = 1.0 - 0.8
- Queries with: nesting level = 2, number of maps = 15
  - cost1: no optimization =2.41E9
  - cost2: only QEG transformation=2.25E9
  - cost3: only Multi-Level Filter/Refinement =1.89E9
  - cost4: full optimization =1.86E9
  - improvement by QEG transformation = 1.0 - 0.933
  - improvement by Multi-Level Filter/Refine = 1.0 - 0.783
  - improvement by both = 1.0 - 0.774

- Queries with: nesting level = 4, number of maps = 15  
cost1: no optimization = 2.42E9  
cost2: only QEG transformation = 2.26E9  
cost3: only Multi-Level Filter/Refinement = 1.89E9  
cost4: full optimization = 1.87E9  
improvement by QEG transformation = 1.0 - 0.933  
improvement by Multi-Level Filter/Refine = 1.0 - 0.783  
improvement by both = 1.0 - 0.774
- Queries with: nesting level = 2, number of maps = 20  
cost1: no optimization = 3.60E9  
cost2: only QEG transformation = 3.35E9  
cost3: only Multi-Level Filter/Refinement = 2.77E9  
cost4: full optimization = 2.73E9  
improvement by QEG transformation = 1.0 - 0.930  
improvement by Multi-Level Filter/Refine = 1.0 - 0.77  
improvement by both = 1.0 - 0.759
- Queries with: nesting level = 4, number of maps = 20  
cost1: no optimization = 3.61E9  
cost2: only QEG transformation = 3.36E9  
cost3: only Multi-Level Filter/Refinement = 2.78E9  
cost4: full optimization = 2.74E9  
improvement by QEG transformation = 1.0 - 0.930  
improvement by Multi-Level Filter/Refine = 1.0 - 0.77  
improvement by both = 1.0 - 0.759
- Queries with: nesting level = 2, number of maps = 25  
cost1: no optimization = 4.73E9  
cost2: only QEG transformation = 4.39E9  
cost3: only Multi-Level Filter/Refinement = 3.59E9  
cost4: full optimization = 3.54E9  
improvement by QEG transformation = 1.0 - 0.928  
improvement by Multi-Level Filter/Refine = 1.0 - 0.759  
improvement by both = 1.0 - 0.748

- Queries with: nesting level = 4, number of maps = 25  
cost1: no optimization =4.74E9  
cost2: only QEG transformation=4.4E9  
cost3: only Multi-Level Filter/Refinement =3.6E9  
cost4: full optimization =3.54E9  
improvement by QEG transformation = 1.0 - 0.928  
improvement by Multi-Level Filter/Refine = 1.0 - 0.759  
improvement by both = 1.0 - 0.748

## B.2 Example randomly generated queries and their QEP before and after applying the optimization techniques

We implemented a demo version of our system that prints out the results of each step (from random query generation to the optimized QEP cost estimation).

When reading the results use the following definitions:

nodej: is the vertical index of the nodes on the QEG.

nodei: enumerates the nodes with the same vertical index on the QEG.

nodeType: determines the type of the node, 2 means sampling node and 1 means normal node.

NULL-ELEMENT: means jump one level on the QEG without having a graph element.

- **Random query1: nesting level = 3, maximum number of maps =5**

The QEG: before QEG transformation

nodeType:2 , nodej:1, nodei:1—POINT

nodeType:1 , nodej:2, nodei:1—WINDOW

nodeType:1 , nodej:3, nodei:1—NULL-ELEMENT

The map name:Map0

The QEG: after QEG transformation

nodeType:2 , nodej:1, nodei:1—WINDOW

nodeType:1 , nodej:2, nodei:1—POINT

nodeType:1 , nodej:3, nodei:1—NULL-ELEMENT

The map name:Map0

The QEP: no QEG transformation and no multi-level filter

nodej = 2, nodei = 1 —FILTER-QNODE

nodej = 2, nodei = 1 —REFINE-QNODE

nodej = 1, nodei = 1 —FILTER-QNODE

nodej = 1, nodei = 1 —REFINE-QNODE

The QEP: with QEG transformation but no multi-level filter

nodej = 2 , nodei = 1—FILTER-QNODE

nodej = 2 , nodei = 1—REFINE-QNODE

nodej = 1 , nodei = 1—FILTER-QNODE

nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: without QEG transformation, with multi-level filter

nodej = 2 , nodei = 1—FILTER-QNODE

nodej = 1 , nodei = 1—FILTER-QNODE

From: nodej = 1 , nodei = 1

To: nodej = 2 , nodei = 1—FEEDBACK-QNODE

nodej = 2 , nodei = 1—REFINE-QNODE

nodej = 1 , nodei = 1—FILTER-QNODE

nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: with both QEG transformation and multi-level filter

nodej = 2 , nodei = 1—FILTER-QNODE

nodej = 2 , nodei = 1—REFINE-QNODE

nodej = 1 , nodei = 1—FILTER-QNODE

nodej = 1 , nodei = 1—REFINE-QNODE

cost1: no optimization = 261586.0

cost2: only QEG transformation = 12911.0

cost3: only Multi-Level Filter/Refinement = 15526.0

cost4: full optimization = 12911.0

- **Random query2: nesting level = 4, maximum number of maps =5**

The QEG: before QEG transformation

nodeType:2 , nodej:1 , nodei:1—UNION

nodeType:1 , nodej:2 , nodei:1—WINDOW

nodeType:1 , nodej:3 , nodei:1—SELECT

nodeType:1 , nodej:4 , nodei:1—NULL-ELEMENT  
 The map name:Map0  
 nodeType:1 , nodej:2 , nodei:2—INTERSECT  
 nodeType:2 , nodej:3 , nodei:2—POINT  
 nodeType:1 , nodej:4 , nodei:2—NULL-ELEMENT  
 The map name:Map1  
 nodeType:2 , nodej:3 , nodei:3—JOIN  
 nodeType:2 , nodej:4 , nodei:3—NULL-ELEMENT  
 The map name:Map2  
 nodeType:2 , nodej:4 , nodei:4—NULL-ELEMENT  
 The map name:Map3

The QEG: after QEG transformation  
 nodeType:2 , nodej:1 , nodei:1—UNION  
 nodeType:1 , nodej:2 , nodei:1—SELECT  
 nodeType:1 , nodej:3 , nodei:1—WINDOW  
 nodeType:1 , nodej:4 , nodei:1—NULL-ELEMENT  
 The map name:Map0  
 nodeType:1 , nodej:2 , nodei:2—INTERSECT  
 nodeType:2 , nodej:3 , nodei:2—POINT  
 nodeType:1 , nodej:4 , nodei:2—NULL-ELEMENT  
 The map name:Map1  
 nodeType:2 , nodej:3 , nodei:3—JOIN  
 nodeType:2 , nodej:4 , nodei:3—NULL-ELEMENT  
 The map name:Map2  
 nodeType:2 , nodej:4 , nodei:4—NULL-ELEMENT  
 The map name:Map3

The QEP: no QEG transformation and no multi-level filter  
 nodej = 3 , nodei = 1—FILTER-QNODE  
 nodej = 3 , nodei = 1—REFINE-QNODE  
 nodej = 3 , nodei = 2—FILTER-QNODE  
 nodej = 3 , nodei = 2—REFINE-QNODE  
 nodej = 3 , nodei = 3—FILTER-QNODE  
 nodej = 3 , nodei = 3—REFINE-QNODE  
 nodej = 2 , nodei = 1—FILTER-QNODE  
 nodej = 2 , nodei = 1—REFINE-QNODE  
 nodej = 2 , nodei = 2—FILTER-QNODE  
 nodej = 2 , nodei = 2—REFINE-QNODE  
 nodej = 1 , nodei = 1—FILTER-QNODE

nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: with QEG transformation but no multi-level filter

nodej = 3 , nodei = 1—FILTER-QNODE  
nodej = 3 , nodei = 1—REFINE-QNODE  
nodej = 3 , nodei = 2—FILTER-QNODE  
nodej = 3 , nodei = 2—REFINE-QNODE  
nodej = 3 , nodei = 3—FILTER-QNODE  
nodej = 3 , nodei = 3—REFINE-QNODE  
nodej = 2 , nodei = 1—FILTER-QNODE  
nodej = 2 , nodei = 1—REFINE-QNODE  
nodej = 2 , nodei = 2—FILTER-QNODE  
nodej = 2 , nodei = 2—REFINE-QNODE  
nodej = 1 , nodei = 1—FILTER-QNODE  
nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: without QEG transformation, with multi-level filter

nodej = 3 , nodei = 1—FILTER-QNODE  
nodej = 3 , nodei = 2—FILTER-QNODE  
nodej = 3 , nodei = 2—REFINE-QNODE  
nodej = 3 , nodei = 3—FILTER-QNODE  
nodej = 3 , nodei = 3—REFINE-QNODE  
nodej = 2 , nodei = 1—FILTER-QNODE  
nodej = 2 , nodei = 2—FILTER-QNODE  
nodej = 1 , nodei = 1—FILTER-QNODE  
From: nodej = 1 , nodei = 1  
To: nodej = 3 , nodei = 1—FEEDBACK-QNODE  
nodej = 3 , nodei = 1—REFINE-QNODE  
nodej = 2 , nodei = 1—FILTER-QNODE  
nodej = 2 , nodei = 1—REFINE-QNODE  
From: nodej = 1 , nodei = 1  
To: nodej = 2 , nodei = 2—FEEDBACK-QNODE  
nodej = 2 , nodei = 2—REFINE-QNODE  
nodej = 1 , nodei = 1—FILTER-QNODE  
nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: with both QEG transformation and multi-level filter

nodej = 3 , nodei = 1—FILTER-QNODE  
nodej = 3 , nodei = 2—FILTER-QNODE  
nodej = 3 , nodei = 2—REFINE-QNODE

nodej = 3 , nodei = 3—FILTER-QNODE  
 nodej = 3 , nodei = 3—REFINE-QNODE  
 nodej = 2 , nodei = 1—FILTER-QNODE  
 nodej = 2 , nodei = 2—FILTER-QNODE  
 nodej = 1 , nodei = 1—FILTER-QNODE  
 From: nodej = 1 , nodei = 1  
 To: nodej = 3 , nodei = 1—FEEDBACK-QNODE  
 nodej = 3 , nodei = 1—REFINE-QNODE  
 nodej = 2 , nodei = 1—FILTER-QNODE  
 nodej = 2 , nodei = 1—REFINE-QNODE  
 From: nodej = 1 , nodei = 1  
 To: nodej = 2 , nodei = 2—FEEDBACK-QNODE  
 nodej = 2 , nodei = 2—REFINE-QNODE  
 nodej = 1 , nodei = 1—FILTER-QNODE  
 nodej = 1 , nodei = 1—REFINE-QNODE

cost1: no optimization =652902.0  
 cost2: only QEG transformation=403485.0  
 cost3: only Multi-Level Filter/Refinement =295986.0  
 cost4: full optimization =291101.0

- **Random query3: nesting level = 3, maximum number of maps =10**

The QEG: before QEG transformation  
 nodeType:2 , nodej:1 , nodei:1—WINDOW  
 nodeType:1 , nodej:2 , nodei:1—WINDOW  
 nodeType:1 , nodej:3 , nodei:1—NULL-ELEMENT  
 The map name:Map0

The QEG: after QEG transformation  
 nodeType:2 nodej:1 nodei:1—WINDOW  
 nodeType:1 nodej:2 nodei:1—NULL-ELEMENT  
 The map name:Map0

The QEP: no QEG transformation and no multi-level filter  
 nodej = 2 , nodei = 1—FILTER-QNODE  
 nodej = 2 , nodei = 1—REFINE-QNODE  
 nodej = 1 , nodei = 1—FILTER-QNODE  
 nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: with QEG transformation but no multi-level filter  
nodej = 1 , nodei = 1—FILTER-QNODE  
nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: without QEG transformation, with multi-level filter  
nodej = 2 , nodei = 1—FILTER-QNODE  
nodej = 1 , nodei = 1—FILTER-QNODE  
From: nodej = 1 , nodei = 1  
To: nodej = 2 , nodei = 1—FEEDBACK-QNODE  
nodej = 2 , nodei = 1—REFINE-QNODE  
nodej = 1 , nodei = 1—FILTER-QNODE  
nodej = 1 , nodei = 1—REFINE-QNODE

The QEP: with both QEG transformation and multi-level filter  
nodej = 1 , nodei = 1—FILTER-QNODE  
nodej = 1 , nodei = 1—REFINE-QNODE

cost1: no optimization =663407.0  
cost2: only QEG transformation=413985.0  
cost3: only Multi-Level Filter/Refinement =306227.0  
cost4: full optimization =301601.0

# BIBLIOGRAPHY

- [ABEL, 1983] Abel, D.J. (1983) *A Data Structure and Algorithm Based on a Linear Key for a Rectangle Retrieval Problem* International Journal of Computer Vision, Graphics and Image Processing, 24, 1, P. 1-13, 1983
- [ABERER, 1994] Aberer, K. (1994) *Semantic Query Optimization for Methods in Object-Oriented Database Systems* Sankt Augustin: Gesellschaft fuer Mathematik und Datenverarbeitung, 1994
- [ATTALURI, 1994] Attaluri, G.K. (1994) *An Efficient Expected Cost Algorithm for Dynamic Indexing of Spatial Objects* CASCION: Proceedings of, P. 193-201, 1994
- [BARRERA, 1981] Barrera, R. (1981) *Schema definition and query language for a geographical database system* Comp. Architecture for Pattern Analysis and Image Database Management, P. 250-256, 1981
- [BCS, 1981] The British Computer Society (1981) *Query Languages* Heyden & Son Ltd., 1981
- [BECKER, 1992] Becker, L. (1992) *Rule-Based Optimization and Query Processing in an Extensible Geometric Database system*, ACM Transactions on Database systems, Vol. 17, No. 2, P. 247-303, City, Country, June 1992
- [BECKER et al., 1999] L. Becker, A. Giesen, K.H. Hinrichs, and J. Vahrenhold (1999) *Algorithms for performing polygonal map overlay and spatial join on massive data sets*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 270-85, Hong Kong, July 1999
- [BECKMANN, 1990] Beckmann, N. (1990) *The R\*-tree: An Efficient and Robust Access Methode for Points and Rectangles* Proc. ACM SIGMOD International Conference on Management of Data, P. 322-330, 1990
- [BELL, 1997] Bell, S. (1997) *Discovering Rules in Relational Databases for Semantic Query Optimization* PADD97 Proceedings of the First International Conference on the Practical Application of Knowledge Discovery and Data Mining, P. 79-90, 1997

- [BERMAN, 1977] Berman, R. (1977) *GEO-QUEL: A system for the manipulation and display of geographic data* Computer Graphics, P. 186-191, 11, 2, 1977
- [BERTINO et al., 1999] E. Bertino and Beng Chin Ooi (1999) *The indispensability of dispensable indexes*, IEEE Transactions on Knowledge and Data Engineering, vol.11, no.1, pp. 17-27, Jan.-Feb. 1999
- [BLAKELEY, 1993] Blakeley, J.A. (1993) *Experiences Building the Open OODB Query Optimizer* SIGMOD/5/93/ Washington, DC, USA, P. 287-296
- [BRABEC et al., 1998a] F. Brabec and H. Samet (1998) *The VASCO R-tree JAVA applet*, Visual Database Systems 4 (VDB4). IFIP TC2/WG2.6 Fourth Working Conference on Visual Database Systems 4 (VDB4), pp. 147-53, L'Aquila, Italy, May 1998
- [BRABEC et al., 1998b] F. Brabec and H. Samet (1998) *Visualizing and animating R-trees and spatial operations in spatial databases on the WorldWide Web*, Visual Database Systems 4 (VDB4). IFIP TC2/WG2.6 Fourth Working Conference on Visual Database Systems 4 (VDB4), pp. 123-40, L'Aquila, Italy, May 1998
- [BRINKHOFF, 1999] T. Brinkhoff (1999) *Requirements of traffic telematics to spatial databases*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 365-9, Hong Kong, July 1999
- [CALCINELLI, 1994] Calcinelli, D. (1994) *Ciglas, a Visual Language for a Geographical Information System: The User Interface* Journal of Visual Language and Computing, Vol. 5, Iss. 2, P. 113-32, June 1994
- [CHANG, 1981] Chang, N. S. (1981) *Picture query language for pictorial database systems* IEEE Computer, P. 23-33, 14, 11, 1981
- [CHEN et al., 1999] J.K. Chen and Y.H. Chin (1999) *A concurrency control algorithm for nearest neighbor query*, Information Sciences, vol.114, no.1-4, pp. 187-204, March 1999
- [CODE, 1990] Codd, E.F. (1990) *The Relational Model for Database Management Version 2*. Addison-Wesley, 1990
- [COORS et al., 1998] V. Coors and V. Jung (1998) *Using VRML as an interface to the 3D data warehouse*, Proceedings. VRML 98 Third Symposium on the Virtual Reality Modeling Language, pp. 121-7, 139-40, Monterey, CA, USA, Feb. 1998
- [CORRAL et al., 1999] A. Corral, M. Vassilakopoulos, and Y. Manolopoulos (1999) *Algorithms for joining R-trees and linear region quadrees*, Advances in Spatial

- Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 251-69, Hong Kong, July 1999
- [COSTAGLIOLA, 1995] Costagliola, G. (1995) *GISQL - A Query Language Interpreter for Geographical Information Systems* Proceedings of The Third IFIP 2.6 Working Conference on Visual, pp. 275-86, 1995
- [CRANSTON et al., 1999] C.B. Cranston, F. Brabec, G.R. Hjaltason, D. Nebert, and H. Samet (1999) *Adding an interoperable server interface to a spatial database: implementation experiences with OpenMap*, Interoperating Geographic Information Systems. Second International Conference, INTEROP'99. Proceedings (Lecture Notes in Computer Science Vol.1580), pp. 115-28, Zurich, Switzerland, March 1999
- [DATE, 1995] Date, C.J. (1990) *An Introduction to Database Systems* Addison-Wesley Publishing Company, Sixth Edition, 1990
- [DE FELICE, 1992] Di Felice, P. (1992) *Towards a Standard for SQL-Based Spatial Query Language* Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing, P. 184-9, 1992
- [DEVOGELE et al., 1998] T. Devogele, C. Parent, and S. Spaccapietra (1998) *On spatial database integration*, International Journal of Geographical Information Science, vol.12, no.4, pp. 335-52, June 1998
- [DUMORTIER et al., 1997] F. Dumortier, M. Gyssens, and L. Vandeurzen (1997) *On the decidability of semi-linearity for semi-algebraic sets and its implications for spatial databases*, Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1997, pp. 68-77, Tucson, AZ, USA, May 1997
- [EGENHOFER, 1994] Egenhofer, M. J. (1994) *Spatial SQL: A Query and Presentation Language* IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 1, Feb. 1994
- [ERWIG et al., 1997] M. Erwig and M. Schneider (1997) *Partition and conquer [spatial data types]*, Spatial Information Theory, A Theoretical Basis for GIS. International Conference COSIT '97 Proceedings, pp. 389-407, Laurel Highlands, PA, USA, Oct. 1997
- [ESTER et al., 1998] M. Ester, H.-P. Kriegel, J. Sander, and Wimmer (1998) *Incremental clustering for mining in a data warehousing environment*, Proceedings of the Twenty-Fourth International Conference on Very-Large Databases, pp. 323-33, New York, NY, USA, Aug. 1998

- [FINKEL, 1974] Finkel, R. A. (1974) *Quad Trees: A data structure for retrieval on composite keys* Acta informatica, no.4, pp. 1-9, 1974
- [FLEWELLING et al., 1999] D.M. Flewelling and M.J. Egenhofer (1999) *Using digital spatial archives effectively*, International Journal of Geographical Information Science, vol.13, no.1, pp. 1-8, Jan.-Feb. 1999
- [FORLIZZI et al., 1998] L. Forlizzi and E. Nardelli (1998) *Some results on the modelling of spatial data*, SOFSEM '98: Theory and Practice of Informatics. 25th Conference on Current Trends in Theory and Practice of Informatics. Proceedings, pp. 332-43, Jasna, Slovakia, Nov. 1998
- [FRANK, 1982] Frank, A. (1982) *Mapquery: Database query language for retrieval of geometric data and their graphical representation* Computer Graphics, P. 199-270, 16, 3, 1982
- [FRANK, 1991] Frank, A. U. (1991) *Language Issues for GIS* In: Geographical Information Systems Principles and Applications, Vol. 1, Longman Scientific & Technical, 1991
- [FREYTAG, 1994] Freytag, J. C. (1994) *Query Processing for Advanced Database systems* Morgan Kaufmann Publishers, 1994
- [FRIAS, 1996] Frias, M. F. (1996) *Semantic Optimization of Queries in Deductive Object-Oriented Database* Advances in Databases and Information Systems. Proceedings of the Second International Workshop on Advances in Databases and Information Systems (ADBIS'95), P. 55-72, 1996
- [FUSSELL, 1981] Fussell, D. (1981) *Deadlock Removal Using Partial Rollback in Database Systems* Proc. ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, P. 65-73, 1981
- [GALINDO, 1997] Galindo-Legaria, C. (1997) *Outerjoin Simplification and Reordering for Query Optimization* ACM Transaction on database systems, P. 43-74, Iss 1, Vol 22, March 1997
- [GUNTHER, 1999] O. Gunther (1999) *Looking both ways: SSD 1999+or-10*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 12-15, Hong Kong, July 1999
- [GUTTMAN, 1984] Guttman, A. (1984) *R-trees: A Dynamic Index Structure for Spatial Searching* Proc. ACM SIGMOD International Conference on Management of Data, Boston, MA, P. 47-57, 1984

- [GOLSHANI, 1992] Golshani, F. (1992) *Design and Specification of EVA: a language for multimedia database systems* Proceedings of DEXA 92. Database and Expert Systems Applications, P. 356-62, 1992
- [GOLSHANI, 1994] Golshani, F. (1994) *Retrieval and delivery of information in multimedia database systems* Information and software technology, Vol. 36, No. 4, P. 235-42, April 1994
- [GONZALES, 2000] M.L. Gonzales (2000) *Seeking spatial intelligence*, Intelligent Enterprise, vol.3, no.2, pp. 28-30, 34-5, 37, Jan. 2000
- [GRAEFE, 1987] Graefe, G. (1987) *The EXODUS Optimizer Generator* Proc. ACM SIGMOD Conference, CA, P. 160-72, May 1987
- [GRUMBACH et al., 1998] S. Grumbach, P. Rigaux, and L. Segoufin (1998) *The DEDALE system for complex spatial queries*, SIGMOD Record, vol.27, no.2, pp. 213-24, June 1998
- [GYSENS et al., 1997] M. Gysens, J. Van den Bussche, and D. Van Gucht (1997) *Complete geometrical query languages*, Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1997, pp. 62-7, Tucson, AZ, USA, May 1997
- [HAZILACOS et al., 1998] T. Hadzilacos and N. Tryfona (1998) *Evaluation of database modeling methods for geographic information systems*, Australian Journal of Information Systems, vol.6, no.1 pp.15-26, Sept. 1998
- [HEE, 1997] Kim Yang Hee (1997) *Two levels of spatial data modeling for an object-oriented spatial database system*, OOIS'97. 1997 International Conference on Object Oriented Information Systems. Proceedings, pp. 397-407, Brisbane, Qld., Australia, Nov. 1997
- [HILLMAN, 1997] R. Hillman (1997) *GIS-based innovations for modeling public transport accessibility*, Geographic Information - Exploiting the Benefits. Proceedings of the AGI'97 Conference, pp. 1-6, Birmingham, UK, Oct. 1997
- [HJALTASON et al., 1998] G.R. Hjaltason and H. Samet (1998) *Incremental distance join algorithms for spatial databases*, SIGMOD Record, vol.27, no.2, pp. 237-48, June 1998
- [HJALTASON et al., 1999] G.R. Hjaltason and H. Samet (1999) *Distance browsing in spatial databases*, ACM Transactions on Database Systems, vol.24, no.2, pp. 265-318, June 1999

- [HO-HYUN et al., 1999] Park Ho-Hyun, Lee Chan-Gun, Lee Yong-Ju, and Chung Chin-Wan (1999) *Early separation of filter and refinement steps in spatial query optimization*, Proceedings. 6th International Conference on Advanced Systems for Advanced Applications, pp. 161-8, Hsinchu, Taiwan, April 1999
- [HWANG, 1994] Hwang, B. (1994) *Spatial Query Processing in Geographic Database Systems* Proceedings of the 20th EUROMICRO Conference. EURONMICRO 94. System Architecture and Integration, P. 53-60, 1994
- [IBRAHIM, 1995] Ibrahim, A. (1995) *Indexing and Retrieving Point and Region Objects* SPIE, Vol. 2670, P. 321-36, 1995
- [IBRAHIM et al., 1996] A. Ibrahim and F. Fotouhi (1996) *Efficient processing of spatial selection and join in databases*, Proceedings of the Fourteenth International Conference on Applied Informatics, pp. 265-7, Innsbruck, Austria, Feb. 1996
- [IONNIDIS, 1996] Ionnidis, Y.E. (1996) *Query Optimization* ACM Computing Surveys, Vol. 28, Iss. 1, P. 121-3, March 1996
- [ISO/TC 211] ISO/TC 211 Work program drafts
- [JARKE, 1984] Jarke, M. (1984) *Query Optimization in Database Systems* Computer surveys, P. 110-152, Vol. 16, No. 2, June 1984
- [JARKE, 1989a] Jarke, M. (1989) *Query Transformation* In: *Query Optimization in KBMS*, KRR Technical Reports, P. 21-9, April 1989
- [JARKE, 1989b] Jarke, M. (1989) *A Framework for Choosing a Database Query Language* Readings in Artificial Intelligence and Databases, Morgan Kaufmann Publishers, P. 363-75, 1989
- [JOSEPH, 1988] Joseph, T. (1988) *PICQUERY: A high level query language for pictorial database management* IEEE Trans. on Software Eng., P.630-638, 14, 15, 1988
- [JUNGERT, 1993] Jungert, E. (1993) *Graqula - A Visual Information-flow Query Language for a Geographical Information System* Journal of Visual Language and Computing, Vol. 4, Iss. 4, P. 383-401, Dec. 1993
- [KLINGER, 1971] Klinger, A. (1971) *Patterns and search statistics* In: *Optimizing Methods in Statistics*, Academic Press, New York 1971
- [KRIEGEL, 1988] Kriegel, H. (1988) *PLOP-Hashing: A grid file without directory* IEEE 4th International Conference on Data Engineering, L.A., P. 369-376, 1988
- [KRIEGEL et al., 1998] H.-P. Kriegel and T. Seidel (1998) *Approximation-based similarity search for 3-D surface segments*, GeoInformatica, vol.2, no.2, pp. 113-47, June 1998

- [LAKSHMI et al., 1998] S. Lakshmi and Zhou Shaoyu (1998) *Selectivity estimation in extensible databases-a neural network approach*, Proceedings of the Twenty-Fourth International Conference on Very-Large Databases, pp. 623-7, New York, NY, USA, Aug. 1998
- [LAURINI, 1992] Laurini, R. (1992) *Fundamentals of Spatial Information System* Academic Press, 1992
- [LAURINI, 1998] R. Laurini (1998) *Spatial multi-database topological continuity and indexing: a step towards seamless GIS data*, International Journal of Geographical Information Science, vol.12, no.4, pp. 373-402, June 1998
- [LAXTON, 1996] Laxton, J. L. (1996) *The Design and Implementation of A Spatial Database for Production of Geological Maps*, Computers & Geosciences, Vol. 22, No. 7, P. 723-733.
- [LAZAR, 1998] B. Lazar (1998) *Break through spatial data translation obstacles*, GIS World, vol.11, no.6, pp. 48-52, June 1998
- [LEE, 1995] Lee, Y. C. (1995) *An iconic query language for topological relationships in GIS* International Journal of Geographical Information Systems, P. 25-46, vol.9, Iss 1, Jan.-Feb. 1995
- [LEVY, 1995] Levy, A. Y. (1995) *Semantic Query Optimization in Datalog Programs* Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on the Principles of Database Systems. PODS 1995, P. 163-73, 1995
- [LU et al., 1995] W. Lu and J. Han (1995) *Query evaluation and optimization in deductive and object-oriented spatial databases*, Information and Software Technology, vol.37, no.3, pp. 131-143, 1995
- [MARTYNOV, 1996] Martynov, M. G. (1996) *Spatial Joins and R-trees* Advances in Databases and Information Systems. Proceedings of the Second International Workshop on Advances in Databases and Information Systems (ADBIS'95), P. 295-304, 1996
- [MATSUYAMA, 1984] Matsuyama, T. (1984) *A File Organization for Geographic Information Systems Based on Spatial Proximity* International Journal of Computer Vision, Graphics and Image Processing, 26, 3, P. 303-318, 1984
- [MCKENNA, 1993] McKenna, W. J. (1993) *Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator* University of Colorado, 1993
- [MELTON, 1995] Melton, J. (1995) *Accommodating SQL3 and ODMG Project: ISO/IEC JTC1.21.3.3 (ISO/IEC JTC1 SC21 WG3 Database Languages)*, May 1995

- [NIGAM, 1997] Nigam, S. (1997) *A Semantic Query Optimization Algorithm for Object-Oriented Databases* Constraint Databases and Applications. Second International Workshop on Constraint Database systems, CDB '97. CP '96 Workshop on Constraints and Databases. Selected Papers, P. 329-43, Cambridge, MA, USA, Aug. 1997
- [OAKLEY, 1994] Oakley, J. (1994) *A Database Management System for Vision Applications* Proceedings of the 5th British Machine Vision Conference, P. 629-39, Vol. 2, 1994
- [OOI, 1991] Ooi, B. C. (1991) *Efficient Query Processing in Geographic Information Systems* Springer-Verlag, 1991
- [ORENSTEIN, 1988] Orenstein, J. A. (1988) *PROBE Spatial data modeling and query processing in an image database application* IEEE Trans. on Software Eng., P. 611-629, I4, 5, 1988
- [OVERMARS, 1982] Overmars, M. H. (1982) *Dynamic multi-dimensional data structures based on quad- and KD- trees* Acta Information, 17, p. 267-285, 1982
- [PAPADIAS et al., 1998] D. Papadias, N. Karacapilidis, and D. Arkoumanis (1999) *Processing fuzzy spatial queries: A configuration similarity approach*, International Journal of Geographical Information Science, vol.13, no.2, pp. 93-118, March 1999
- [PAPADIMITRIOU et al., 1998] C.H. Papadimitriou, D. Suciu, and V. Vianu (1999) *Topological queries in spatial databases*, Journal of Computer and System Sciences, vol.58, no.1, pp. 29-53, Feb. 1999
- [PAPADOPOULOS et al., 1997] A. Papadopoulos and Y. Manolopoulos (1997) *Nearest neighbor queries in shared-nothing environments*, GeoInformatica, vol.1, no.4, pp.369-92, Dec. 1997
- [PAPADOPOULOS et al., 1999] A. Papadopoulos, P. Rigaux, and M. Scholl (1999) *A performance evaluation of spatial join processing strategies*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 286-307, Hong Kong, July 1999
- [PAREDAENS, 1995] Paredaens, J. (1995) *Spatial Databases, The Final Frontier* Proceedings of Database Theory-ICDT 95. 5th International Conference, P. 14-32
- [PAREDAENS et al., 1998] J. Paredaens and B. Kuijpers (1998) *Data models and query languages for spatial databases*, Data & Knowledge Engineering, vol.25, no.1-2, pp. 29-53, March 1998

- [PARK et al., 1999] Ho-Hyun Park, Guang-Ho Cha, and Chin-Wan Chung (1999) *Multi-way spatial joins using R-trees: methodology and performance evaluation*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 229-50, Hong Kong, July 1999
- [PFOSER et al., 1999] D. Pfoser and C.S. Jensen (1999) *Capturing the uncertainty of moving-object representations*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 111-31, Hong Kong, July 1999
- [RAVADA et al., 1999] S. Ravada and J. Sharma (1999) *Oracle8i Spatial: experiences with extensible databases*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 355-9, Hong Kong, July 1999
- [RISHE 95] Rishe, N. (1995) *Florida International University High Performance Database Research Center SIGMOD Record*, Vol. 24, Iss. 3, P. 71-6, Sept. 1995
- [ROUSOPOILOS, 1985] Roussopoulos, N. (1985) *Direct spatial search on pictorial databases using packed R-trees* Proc. ACM SIGMOD Int. Conf. on Management of Data, Austin, Texas, P. 17-31, 1985
- [SAMET, 1984] Samet H. (1984) *The Quadtree and Related Hierarchical Data Structures* ACM Comp. Survey 16, P. 187-26, 1984
- [SAMET, 1991a] Samet, H. (1991) *Applications of Spatial Data Structures* ADDISON-WESLEY PUBLISHING COMPANY
- [SAMET, 1991b] Samet, H. (1991) *The Design and Analysis of Spatial Data Structures* ADDISON-WESLEY PUBLISHING COMPANY
- [SAMET, 1995] Samet, H. (1995) *General Research Issues in Multimedia Database Systems* ACM Computing Surveys, P. 630-632, Vol. 27, No. 4, December 1995
- [SANDER et al., 98] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu (1998) *Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications*, Data Mining and Knowledge Discovery, vol.2, no.2, pp.169-94, 1998
- [SCHNEIDER, 1999] M. Schneider (1999) *Uncertainty management for spatial data in databases: fuzzy spatial data types*, Advances in Spatial Databases. 6th International Symposium, SSD'99. Proceedings (Lecture Notes in Computer Science Vol.1651), pp. 330-51, Hong Kong, July 1999
- [SEAI, 1990] SEAI Technical Publications (1990) *Geographic Information Systems: An Assessment of Technology, Applications and products* Vol. 3, 1990

- [SEGOUFIN et al., 1998] L. Segoufin and V. Vianu (1998) *Querying spatial databases via topological invariants*, Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS 1998, pp. 89-98, Seattle, WA, USA, June 1998
- [SELLIS, 1991] Sellis, T. K. (1991) *Query Optimization for Nontraditional Database Applications* IEEE Transactions on Software Engineering, P. 77-86, Vol. 17, No. 1, Jan. 1991
- [SELLIS et al., 1997] T. Sellis, N. Roussopoulos, and C. Faloutsos (1997) *Multidimensional access methods: trees have grown everywhere*, Proceedings of the Twenty-Third International Conference on Very Large Databases, pp. 13-14, Athens, Greece, Aug. 1997
- [SHEIKHOESLAMI et al., 1998] G. Sheikholeslami, S. Chatterjee, and A. Zhang (1998) *WaveCluster: a multi-resolution clustering approach for very large spatial databases*, Proceedings of the Twenty-Fourth International Conference on Very-Large Databases, pp. 428-39, New York, NY, USA, Aug. 1998
- [SHEIKHOESLAMI et al., 2000] G. Sheikholeslami, S. Chatterjee, and Zhang Aidong (2000) *WaveCluster: a wavelet-based clustering approach for spatial data in very large databases*, VLDB Journal, vol.8, no.3-4, pp. 289-304, Feb. 2000
- [SHEKHAR et al., 1999] S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, Liu Xuan, and Lu Chang-Tien Lu (1999) *Spatial databases-accomplishments and research needs*, IEEE Transactions on Knowledge and Data Engineering, vol.11, no.1, pp. 45-55, Jan.-Feb. 1999
- [SMITH, 1998] R. Smith (1998) *Web-enabled GIS - an open component based approach*, AGI Conference at GIS 98. Profiting from Collaboration, pp. 17-21, Birmingham, UK, Oct. 1998
- [SONNEN, 2000] D. Sonnen (2000) *Spatial information: out of the basement*, Intelligent Enterprise, vol.3, no.2, pp. 38-43, Jan. 2000
- [SQL Home Page] SQL Standard Home Page <http://www.jcc.com/sqlstnd.html>
- [STONEBRAKER, 1987] Stonebraker, M. (1987) *The POSTGRES data model* Proc. 13th Int. conf. on very large databases, P. 83-96, 1987
- [THEODORIDIS, 1995] Theodoridis, Y. (1995) *Range Queries Involving Spatial Relations: A Performance Analysis* Spatial Information Theory. A Theoretical Basis for GIS. Proceedings of International Conference COSIT 95, P. 537-51, 1995

- [THEODORIDIS et al., 1998] Y. Theodoridis, E. Stefanakis, and T. Sellis (1998) *Cost models for join queries in spatial databases*, Proceedings. 14th International Conference on Data Engineering, pp. 476-83, Orlando, FL, USA, Feb. 1998
- [THERIAULT, 1996] Thériault, S. and Kerh erve, B. (1996) *Survey on Existing Spatial Indexing Techniques* CRIM/IIT, Apr. 1996
- [ULLMAN, 1989] Ullman, J.D. (1989) *Principles of Database and Knowledge-Base systems*, Computer Science Press,
- [VALDURIEZ, 1984] Valduriez, P. (1984) *Join and Semijoin Algorithms for a Multiprocessor Database Machine* ACM Transaction on Database Systems, Vol. 9, no. 1, P. 133-61, 1984
- [VANDEURZEN 98] L. Vandeurzen, M. Gyssens, and D. Van Gucht (1998) *An expressive language for linear spatial database queries*, Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS 1998, pp. 109-18, Seattle, WA, USA, June 1998
- [VASSILAKOPOULOS, 1995] Vassilakopoulos, M. (1995) *Dynamic Inverted Quadtree: A Structure for Pictorial Databases* Information systems, P. 483-500, Vol. 20, No. 6, 1995
- [WANG, 1990] Whang, K. (1990) *Query Optimization in a Memory-Resident Domain Relational Calculus Database System* ACM Transactions on Database Systems, P. 67-95, Vol. 15, No. 1, March 1990
- [WEI et al., 1997] Wang Wei, Yang Jiong, and R. Muntz (1997) *STING: a statistical information grid approach to spatial data mining*, Proceedings of the Twenty-Third International Conference on Very Large Databases, pp. 186-95, Athens, Greece, Aug. 1997
- [WESSEL et al., 1998] M. Wessel and V. Haarslev (1998) *VISCO: bringing visual spatial querying to reality*, Proceedings. 1998 IEEE Symposium on Visual Languages, pp. 170-7, Halifax, NS, Canada, Sept. 1998
- [XU et al., 1999] Xiaowei Xu, J. Jager, and H.-P. Kriegel (1999) *A fast parallel clustering algorithm for large spatial databases*, Data Mining and Knowledge Discovery, vol.3, no.3, pp. 263-90, 1999
- [YUN-WU et al., 1998] Huang Yun-Wu, M. Jones, and E.A. Rundensteiner (1998) *Symbolic Intersect Detection: a method for improving spatial intersect joins*, GeoInformatica, vol.2, no.2, pp. 149-74, June 1998

- [ZHAO, 1994] Zhao, J.L. (1994) *Spatial Data Traversal in Road Map Databases: A Graph Indexing Approach* CIKM 94 Proceedings of The Third International Conference on Information and Knowledge Management, P. , 1994
- [ZIMBRAO et al., 1998] G. Zimbrão and J. Moreira de Souza (1998) *A raster approximation for the processing of spatial joins*, Proceedings of the Twenty-Fourth International Conference on Very-Large Databases, pp. 558-69, New York, NY, USA, Aug. 1998